GGTP, a General Gaming Text Protocol

Michel Quenault and **Tristan Cazenave**
LAMSADE
Université Paris-Dauphine, France
miq75@free.fr

Abstract

This article describes GGTP, a new general gaming text protocol enabling to play and record plays of many kinds of games. These games range from classic complete information board games such as Chess to incomplete information and chance games such as Bridge. In theory, even real time strategy games could be managed, but theses games currently imply practical limits.

1 Text Protocol for games

Many games are played around the world. For most of them, players training and development of competitions require a way to describe precisely specific plays of the games. So many different play systems have been developed, each time with specific data structures for a specific game.

In artificial intelligence, games have always been a good domain for the exploration of the performance of decision methods. Many separated researches are and have been done all around the world on such themas and many researchers have to define games engines. One more time, with the same restrictions than for human players, the specialisation in a specific rule.

Fewer researches focusses on managing general gaming engines which are designed to be able to play many games. In such problems games engines have to be separated from the rules definitions so they are decomposed in few distinct programs with their own independents goals. To work together and to manage the plays of a selected game rule, these separate components have to use some protocoled communications. These protocoled communications are inherited from the description language used by human players to manage human training and human competitions.

Let's first focus on what are precisely the different components and goals of such a protocol, then we comment on variety of games and describe some existing protocols.

1.1 Requirements of a text protocol for games

A play description protocol has to define instructions for each step of a play.

First, initialisation of the play. A general gaming protocol has to allow to select the different following elements to prepare a play:

- the used rule,
- the engines used to play,
- any specific initialisation data from the rule (such as the first player to play),
- any parameters specific to the rules or the engines selected.

Second, and as many times as the rules define it, the protocol has to specify a way to describe the moves selected by the engines. When moves have to be validated by the rule such as in Kriegspiel, Phantom Go, Mah Jong or when multiple players can play at the same turn, moves have to be validated both by the engine and the referee.

Third, at the end of the play, the protocol has to manage final information such as the winners, losers, or null players and optionally scoring data for players if the rule specify any as in card games.

Possibly, some control commands such as the ability to navigate between steps of the play (undoing or redoing moves, or selecting different moves) would be useful to define too. Perhaps some ability to manage subtrees of plays, to show for example some specific lines of play that a teacher thinks important.

Another kind of commands could be considered, those who enable to manage tournaments. [1]

Such protocol can be used to follow and record any play of any game. It would be nice to use a text style protocol to allow easy reading of the steps of the play by human analysts. This format would have the advantage to allow humans to interfere with the evolution of the play, with possibly some appropriate commands to override the automatic players choices.

Some of these commands are sent by automatic engines, some are sent by the rule, and some are sent by the humans. Obviously, the protocol has to define who can send which kind of command too.

1.2 Variety of games

Many examples of games have been studied individually in artificial intelligence researches. We can cite as examples Solitaire [Beasley, 2008], Sudoku [Geem, 2007], Sokoban

[1] In a way, a competition is a kind of super rule. So a correctly defined protocol, with an engine able to manage really various games can define a competition as a specific game rule.

[Botea *et al.*, 2002], Rush Hour [Fernau *et al.*, 2003], Checkers [Schaeffer *et al.*, 2007; Khan *et al.*, 2008], Hex [Henderson and Hayward, 2008], Chess [Fernández and Salmerón, 2008], Shogi [Soeda *et al.*, 2006], Go [Bouzy and Helmstetter, 2003], Chinese Checkers [Cooper *et al.*, 2004], Backgammon [Azaria and Sipper, 2005], Can't Stop [Glenn *et al.*, 2008], Bridge [Galatti *et al.*, 2005], Skat [Kupferschmid and Helmert, 2006], Poker [Gilpin and Sandholm, 2007], Phantom Go [Cazenave, 2006], Kriegspiel [Ciancarini and Favini, 2007], Mah Jong [Iida *et al.*, 2004], some wargames [Santana *et al.*, 2004] and some real time strategy games [Ontañón *et al.*, 2007].

These games, except for a part of the wargames and the real time strategy games, are all discrete and finites games. Discrete games are games with no continuity of any kind. Boards must be discrete, alternate of players moves must be discrete, and so on. Finite games are games with no infinity of any kind. Boards must be finite, list of moves must be finite, and so on.

Indeed, all of them have different specificities, Here is a short list of critic ones:

- Number of players:

  Some of studied games are one player games. These puzzle games generally have their own solving class of methods.

  Some other studied games are two player games. These games use different methods from puzzle games because they have to consider having an opponent whose goals are generally in opposition with the player's goals.

  The last part are games with more than 2 players. In these games, methods from two players games could be reused but they generally miss some point, because there are specific parameters to deal with. (For example the possibility of alliances between players or the definition of some priority for the players such the choice to be the only one to win or not.) This kind of games generally require special algorithms for these problems and this make them different to program.

  Despite theses differences, this distinction between games do not denote specific problem for the point of view of a general gaming protocol.

- Randomness:

  Some board games, such as backgammon, (and furthermore some non board games) may have to deal with randomness.

  As for the number of players, this particularity of games do not implies any problem for a general gaming protocol.

- Incomplete information:

  There are two kinds of incomplete information games. The incomplete information can be about the possible moves of other players (which depend on their hands in card games) or about the possible moves of players itself (as in Phantom Go).

  If the first case doesn't have any impact on a general game protocol between engines players and the rule, the second case implies that such a protocol must deal with the possibility that the rule refuses a move that the engine player considers as legal, with maybe the reason why. This ability would not be complex to add to such a protocol.

- Simultaneous moves multiple players games:

  In most of games, the players alternate the moves. In some games, like Mah Jong, their may be case where the normal succession of the players can be broken, and where multiple player can play at the same time. There are specific priority rule to deal in these cases.

  In some variants of Uno card games, or in Jungle Speed, the graph of succession of players moves can be very complex. It happens many times when playing these games that multiple players can play, and the fastest one has the point.

  These games are still discrete ones because there is always a rule to define which of the different player moves is the valid one and what is the consequence on the next moves. Indeed, for a general gaming protocol, this falls back to the case of incomplete information games when the rules has to refuse a move to a player. So these kind of games do not represent a difficult problem for such a protocol.

We now present some existing general gaming protocols.

1.3 Known languages

Some different protocols have already been implemented to describe multiple games.

We can cite researches from Barney Pell which has describes with Metagame a general game model dealing with chess symmetric variants [Pell, 1994], the Stanford logic group which has defined his own general gaming protocol for competitions [Love *et al.*, 2008]. The current best General Game Playing agents use Monte-Carlo methods [Finnsson and Björnsson, 2008], even if alternative architecture are also used [Schiffel and Thielscher, 2007]. Arno Hollosi has defined a protocol based on textual commands [Hollosi, 2006] and another based on XML style [Hollosi, 2002]. Michael Buro and Igor Ðurđanović use NECI's generic game server, based on textual commands [Buro and Ðurđanović, 2002].... A widely used protocol to make Go and Hex programs play against each other and play using an interface is the Go Text Protocol (GTP) [Farnebäck, 2002].

All these protocols are limited to complete information games and actually only relatively few games are defined with each of them. Thus, they may contain rule specific commands such as Komi or Handicap in SGF which refers to specific (here: Go) game properties.

These protocols were developed as research tools for general gaming. They often come with rule description tools and independent engines players adding capabilities. The rule description is generally made with logical languages to allow easy analyse of the rule for the players engines.

Other tools are designed for specific interactions with an engine. For example general tools to communicate with engines have been developed for analyzing search trees [Ísleifsdóttir and Björnsson, 2008].

On the other hand, the commercial Zillions of games engines [Lefler and Mallett, 2002] use a protocol which can describe a very big number (more than 1500) of rules, including one player games or more than two player ones. The rule description in Zillions of games is logical too, but until recently, their would be no way to describe it's own engine. Thus, this application is still limited to complete information and non random games.

Either, we had to talk about some protocols which focus on totally different kind of games than complete information ones. For example, the ORTS engine would provide similar tools to play to real time strategy games [Buro, 2002]. This time, the protocol is very specific to RTS games and could not be used for other ones. So, we are confronted to the same problem than existent complete information general gaming protocols, which is their specialisation to very restricted kinds of games.

2 GGTP

We have seen that almost all of the engines are limited to complete information games, and to two players games in general gaming oriented researches. We have seen that many other kinds of games are studied in artificial intelligence. So, we propose here to describe the first steps of a general gaming protocol that would manage both complete information games and incomplete information ones. We focus on making it usable for a maximum of these games, so we intend to develop it to include random games, any number of players games and multiple simultaneous players moves games. The limit we fix to our protocol is the possibility to use it with any discrete and finite game. Here is our proposition.

This protocol has to allow an application using it to be a totally automated engine usable in general gaming researches.

We use a textual protocol with one line per command. Commands and each arguments are separated with space or tabulations. An earlier add ability for the protocol was the possibility to add spaces in moves names, because it was restrictive for rules to define only non spaced ones. So the move names or list of moves names are always the last argument and spaces are no more considered as separators when the maximum number of arguments for a command is reached. Thus, due to the wide variety of possible moves description in games, the line length was defined as unlimited. This way, the protocol can manage spaces in move names and complex multiple part move on some games could be used easily.

2.1 Supervisor part

To develop a general gaming system, their are a few different items to consider. The first is a way to describe the rules and to manage a play of the described rule, the second is a way to define general players engines that can be connected to the part that manages the rule, and the last is the protocol that allows communications between the elements cited below and with a human supervisor of the play. We now refer to these components as the Rules, the Engines and the Protocol. We add the Supervisor component to represent the human regulating them.

The part of communication from Rules to Supervisor is the output of the main application. For each command the Supervisor can send, there are three kinds of answers:

- An answer relative to the query, such as the version number of the Rules for the `get_version` command, preceded with the "`=`" symbol.
- A validation mark ("`Done`") if the command is not interrogative.
- An error message if the command is incorrectly formulated, preceded with the "`?`" symbol. The message tells to use "`list_commands`" if the command is unknown and describe the arguments needed by the command in the other case.

We add to these returns some options relatives to the Rules output. The options we had defined are "`show_command`", "`show_when_play`", "`show_possible_moves`", "`show_results`" and "`show_table`" (which refer to the complete set of the physical elements of the game). They all can take the two values "`on`" or "`off`", defining this way if Rules informations are displayed or not on the screen after each step of the play. These informations are preceded with the "`->`" symbol when they are shown.

A last option is relative to the Rules management, the "`genmove`" option which determinate if Engines are playing as soon as the Rules allows them to do it or only on Supervisor request.

With these elements we have described the first part of the Protocol, which is the way the Rules communicate with the Supervisor. The way the Supervisor communicate with the Rules is described in the Table 1.

The first command shows the program version, the second one leaves it.

The commands 3 to 5 are about file manipulations. They allow to use files as convenient ways to store plays. In a play file, only a part of the commands executed since the beginning of application, are stored. These commands are marked with a "✓" in the table. They mainly are commands that effectively do an alteration on the play or the play initialisation. Every made move is marked as if is was selected by the command "`force_play`". Thus, a play could be stored in the middle of its unfolding and be restored later. Command "`force_play`" is not registered itself but the move which has been validated by the rule is. The support for storage of multiple branches has not been defined yet.

Commands 6 and 7 are queries about the possible commands list.

Commands from 8 to 11 are used to set the parameters defining the format of outputs for the Supervisor we have seen above.

Commands from 12 to 18 concern the current rule which is selected by the engine and it's specific options. One of these options called "`starting_player`" is automatically generated if the rule doesn't define its first player.

While command 19 references the current list of players selected by the rule, the following commands until 22 select the Engines to connect to the players of the current rule. When an Engine is selected this way, it is created and connected to the

1	get_version		
2	quit		
3	load_file	filename	
4	save_file	filename	
5	close_file		
6	list_commands		
7	known_command	command	
8	list_options		
9	get_option	option	
10	list_option_choices	option	
11	set_option	option value	✓
12	list_rules		
13	set_rule	rule	✓
14	get_rule		
15	list_rule_options		
16	get_rule_option	option	
17	list_rule_option_choices	option	
18	set_rule_option	option value	✓
19	list_players		
20	list_engines		
21	set_engine	player engine	✓
22	get_engine	player	
23	list_engine_options	player	
24	get_engine_option	player option	
25	list_engine_option_choices	player option	
26	set_engine_option	player option value	✓
27	start		✓
28	list_possible_moves	[[player] table]	
29	get_result	[[player] table	
30	get_table	[[player] table	
31	get_last_move		
32	list_played_moves		
33	play	player move	
34	force_play	player move	✓
35	genmove		
36	undo		
37	redo		

Table 1: GGTP Supervisor to Rules commands.

Rules with a new bidirectional socket. The Supervisor can set any player of the current game to human. This way, the Supervisor can play against Engines. Today, the Engines are locals and created when they are required by the application, but we intend to implement later a network interface for players. This way, some human or Engines players would have the ability to play through the internet or any network.

Commands from 23 to 26 manage specific options for the Engines.

Then the Supervisor needs the command 27 to initiate the play.

Commands 28 to 32 enable to display informations about the play. The same information would be displayed when activating the correct relative to Rules output or management options. (The optional arguments in command 28 to 30 refer to the point of view of a player and his knowledge about the play, not to the specific to the player possible moves knowns by the Rules. The second argument refers to the number of the simulation space (table) used by the player). These informations are necessary to play when the Supervisor has defined human players.

Commands 33 to 35 are controlling the next moves in the play. They are respectively used to play as a human player, play as the Supervisor and launch the play of an Engine player if the "`genmove`" option is unactivated.

The two last commands enable the supervisor to navigate through the played moves and to test other moves at any step. Today, this would delete the previous branch of moves but in a next version we would like to manage the ability to store the explored tree.

2.2 Engine part

After we have described the Supervisor parts of the Protocol, we focus on the Engine part. Unlike the Supervisor commands, there is no place for syntax errors or malformed commands. No unknown command is admitted and a query requests an immediate answer, except for the move selection which is asynchronous. The Rules to Engines commands are enumerated in Table 2.

38	connect_engine	
39	disconnect_engine	
40	list_options	
41	get_option	option
42	list_option_choices	option
43	set_option	option value
44	start	
45	list_possible_moves	table
46	get_result	table
47	get_table	table
48	have_to_play	[delay]
49	made_move	player move
50	illegal_move	move [any]
51	undone_move	
52	redone_move	

Table 2: GGTP Rules to Engines commands.

Commands 38 and 39 are about connecting the Engine to a player in the current game. They are consequences of the command 21. They initiate the connection (after the Engines has possibly been created) and terminate it (before the Engines are destroyed).

Commands from 40 to 45 are redirections from commands 23 to 28 on the right Engine.

Command 48 is sent whenever the Engine has to select a move. This happens in two cases depending on the "`genmove`" option's value: When the command 35 is called by the Supervisor or when the Rules has computed the last move and possibly the result. The rule may define a delay in which the Engine has to send an answer and pass it as argument of the command.

The commands 49 to 52 are those who actually validate (or not) the chosen move, inform that a move has been undone or redone by the Rules. The Engine has to consider them immediately. Command 50 is used in games such as Phantom Go so that the referee can tell the engine a move is illegal, the second argument can contain any additional information specific to the rule.

Then, the next part of the protocol are the communications from Engines to Rules which is describe with the Table 3.

53	reply	requested data
54	acknowledge	
55	invalid_option_name	
56	invalid_option_value	
57	invalid_table_number	
58	select_move	move

Table 3: GGTP Engines to Rules commands.

The 53 to 57 commands are the different immediate answers that the Engines has to manage. Command 53 is used in any of the commands 40, 41, 42, 45, 46, 47 with the asked result and command 54 to 57 reply to other commands to validate them or refuse their incorrect arguments.

The last command tells to the Rules the selected next possible move. It's an asynchronous reply to command 46.

Commands 45, 46, 47 and 58 show that the Engines have their own representation of the play state. This avoids the need for communications between Rules and Engines in the simulations of possible tree of plays.

With these last items we have finished to describe our GGTP Protocol. Next will come the description of an application using it.

3 Description of the application

With the GGTP protocol we have defined a complete general gaming engine. This engine has been developed to manage all the games that GGTP can support. Without having a too deep description of the way it manages the rule, which (an older version) can be found in [Quenault and Cazenave, June 2007], we shortly describe the way our Rules and Engines separated components interfer with each others and enable general play.

3.1 Rules descriptions

Here is the list of what our Rules uses from GGTP:

- The almost complete commands list. The few exceptions are cited below.
- The complete set of relative to Rules output and management options described above.
- Any specialisation of the Rule via the options commands.
- The players list abilities, including the "`starting_player`" option, based on a graph description of the turn order.
- A machine representation of the physics components of the game (cards, pawns, board...), corresponding to the "table" expression used in command 30.
- The initialisation of the game, applied with the command 27.
- The end of the game with specifying the winning or losing players, used by the command 29.
- The moves that are allowed to the players at each step of the play. The way to identify moves is totally rule specific and only the Rules and possibly the Engines have to attribute to moves names a sense. This is why we consider that for the Protocol point of view, each move description is meaningless and could just be a textual item.

Our way to describe an new rule today is a c++ source compiled with the application and all the Engines. This is a problem for developing an ability to analyse the rules for the engines. That is why GGTP does not implement commands in this way yet.

Both the nature of c++ code of our rules and their abilities to have rule's specialised components made that our compilation tree first nodes are rules objects. So, each of our rules are linked as independent programs. That's why the "`set_rule`" command has not been implemented. In a version based on a logical description of the rule, we guess that this problem would be unwound.

The "`save_file`" command is not implemented yet. There is no storage of current session before the "`start`" command. (But the list of moves is stored and navigation throughout is operational.)

Optional argument for the "`list_possible_moves`" command is implemented only with human players. The commands doesn't refer to the Engines point of view and the corresponding commands in the engine part of the protocol are not yet implemented.

No timing systems are implemented, only the asynchronous nature of the players yet.

3.2 Engines descriptions

We now list what our Engines uses from GGTP:

- The complete command list concerning Engines, with few simplifications.
- Any specialisation of the Engine via the options commands.
- Complete access to simulations spaces (we call them tables) containing all the elements of the games rule in their current state. This enables the engine to manage its own simulations of plays.

Each engine may have any number of simulation spaces (tables). To manage incomplete information games, each table is a complete information view of a possible real table of the play. In any game, when a move is validated, all the tables are updated. If the game is an incomplete information one, the validated move could be illegal on some tables. In such a case, the tables have to operate few alterations (reorder few components as cards in hands) to make the move legal. In incomplete information games, the same manipulation is performed when using the "`force_play`".

Our engine allows a filename argument when the application is started on the command line. This file is automatically read at the beginning of the session. This way, and using correctly some predefined options of the protocol, we could use our application as a totally automated one able to play any defined Engines against any defined Rules.

3.3 Short example

Now that we had expose our application components let's have a look at Figure 1, a short example of a play using it and GGTP.

This example shows two different processes displayed with vertical lines. The continue one is the Supervisor to Rules part of the protocol and the disjoint one is the Rules to Engines part of the protocol (Here, the "`Circle`"'s engine). Whenever the Supervisor part begins with a letter it represents a command send by the Supervisor. In the other cases ("=", "?" or "->") it represents the Rules answers. For the Engines part, the symbol "↩" separates the commands sent by the Rules to the Engines (before the symbol, or if it is omitted) to the commands sent by the Engines to the Rules, which are always answers (after the symbol).

This example of the protocol is totally implemented by our application, with the exception of the "`set_rule`" command.

During the real use of the application only the Supervisor part is printed on screen. We have represented here both parts to illustrate the complete GGTP protocol.

4 Conclusion and future works

Most of the GGTP protocol has been implemented. It can be used for general gaming development purposes. Our main goal is now to complete the associated engines so as to manage incomplete information games using Monte Carlo methods. We would also like to manage games with time limits, incomplete information about the possible players moves and the basics of all kinds of discrete and finite games.

But there are still some lacks in our engine and GGTP protocol:

- Commands to enable engines players to deliberate with other players.
- A way to enable the storage of multiple subtrees of the possible plays graphs.
- Commands to manage the networked connection of general gaming players.
- Commands to manage competition between general gaming players.
- Adding much more games and much more general gaming players.

When all these different points are realized, GGTP would probably be a reference in general game playing protocols. GGTP enables to manage many different kinds of games including incomplete information ones.

References

[Azaria and Sipper, 2005] Yaniv Azaria and Moshe Sipper. Using gp-gammon: Using genetic programming to evolve backgammon players. In Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano I. van Hemert, and Marco Tomassini, editors, *EuroGP*, volume 3447 of *Lecture*

```
set_rule TicTacToe
= Done
set_option show_when_play on
= Done
set_option genmove on
= Done
list_players
= Cross=human, Circle=human
set_rule_option starting_player Cross
= Done
set_enginz Circle Minimax
?  Use get_commands to see commands.
set_engine Circle Minimax
   | connect_engine ↩ acknowledge
= done
set_engine_option Circle Depth 8
   | set_option Depth 8 ↩ acknowledge
= done
start
   | start ↩ acknowledge
= done
play Cross A1
-> Cross plays A1
   | made_move Cross A1 ↩ acknowledge
   | have_to_play
= done
   | ↩ select_move B2
-> Circle plays B2
   | made_move Circle B2 ↩ acknowledge
play Cross A1
?  No move named A1 for player Cross.
get_possible_moves
= cross=B1,C1,A2,C2,A3,B3,C3
set_option get_possible_moves on
= Done
set_option genmove off
= Done
play Cross A2
-> Cross plays A2
   | made_move Cross A2 ↩ acknowledge
-> Circle=B1,C1,C2,A3,B3,C3
= Done
genmove
   | have_to_play
= Done
   | ↩ select_move A3
-> Circle plays A3
   | made_move Circle A3 ↩ acknowledge
-> Cross=B1,C1,C2,B3,C3
undo
-> Undoing a move
   | undone_move ↩ acknowledge
-> Circle=B1,C1,C2,A3,B3,C3
= Done
...
```

Figure 1: GGTP part of sample play.

Notes in Computer Science, pages 132–142. Springer, 2005.

[Beasley, 2008] John D. Beasley. Solitaire: Recent developments. *CoRR*, abs/0811.0851, 2008.

[Botea *et al.*, 2002] Adi Botea, Martin Müller 0003, and Jonathan Schaeffer. Using abstraction for planning in sokoban. In Schaeffer et al. [2003], pages 360–375.

[Bouzy and Helmstetter, 2003] Bruno Bouzy and Bernard Helmstetter. Monte-carlo go developments. In H. Jaap van den Herik, Hiroyuki Iida, and Ernst A. Heinz, editors, *ACG*, volume 263 of *IFIP*, pages 159–174. Kluwer, 2003.

[Buro and Ðurđanović, 2002] Michael Buro and Igor Ðurđanović. An overview of neci's generic game server. 2002. http://www.cs.ualberta.ca/ mburo/ps/ggs.pdf.

[Buro, 2002] Michael Buro. Orts: A hack-free rts game environment. In Schaeffer et al. [2003], pages 280–291.

[Cazenave, 2006] Tristan Cazenave. A phantom-go program. In van den Herik et al. [2006], pages 120–125.

[Ciancarini and Favini, 2007] Paolo Ciancarini and Gian Piero Favini. Representing kriegspiel states with metapositions. In Manuela M. Veloso, editor, *IJCAI*, pages 2450–2455, 2007.

[Cooper *et al.*, 2004] Nicholas Cooper, Aaron Keatley, Maria Dahlquist, Simon Mann, Hannah Slay, Joanne Zucco, Ross Smith, and Bruce H. Thomas. Augmented reality chinese checkers. In *Advances in Computer Entertainment Technology*, pages 117–126. ACM, 2004.

[Farnebäck, 2002] Gunnar Farnebäck. Specification of the go text protocol, version 2. Technical report, http://www.lysator.liu.se/ gunnar/gtp, October 2002.

[Fernández and Salmerón, 2008] Antonio Fernández and Antonio Salmerón. Bayeschess: A computer chess program based on bayesian networks. *Pattern Recognition Letters*, 29(8):1154–1159, 2008.

[Fernau *et al.*, 2003] Henning Fernau, Torben Hagerup, Naomi Nishimura, Prabhakar Ragde, and Klaus Reinhardt. On the parameterized complexity of the generalized rush hour puzzle. In *CCCG*, pages 6–9, 2003.

[Finnsson and Björnsson, 2008] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *AAAI*, pages 259–264, 2008.

[Galatti *et al.*, 2005] John Galatti, Sung-Hyuk Cha, Michael L. Gargano, and Charles C. Tappert. Applying artificial intelligence techniques to problems of incomplete information: Optimizing bidding in the game of bridge. In Hamid R. Arabnia and Rose Joshua, editors, *IC-AI*, pages 385–392. CSREA Press, 2005.

[Geem, 2007] Zong Woo Geem. Harmony search algorithm for solving sudoku. In Bruno Apolloni, Robert J. Howlett, and Lakhmi C. Jain, editors, *KES (1)*, volume 4692 of *Lecture Notes in Computer Science*, pages 371–378. Springer, 2007.

[Gilpin and Sandholm, 2007] Andrew Gilpin and Tuomas Sandholm. Better automated abstraction techniques for imperfect information games, with application to texas hold'em poker. In Edmund H. Durfee, Makoto Yokoo, Michael N. Huhns, and Onn Shehory, editors, *AAMAS*, page 192. IFAAMAS, 2007.

[Glenn *et al.*, 2008] James Glenn, Haw ren Fang, and Clyde P. Kruskal. A retrograde approximation algorithm for multi-player can't stop. In van den Herik et al. [2008], pages 252–263.

[Henderson and Hayward, 2008] Philip Henderson and Ryan B. Hayward. Probing the 4-3-2 edge template in hex. In van den Herik et al. [2008], pages 229–240.

[Hollosi, 2002] Arno Hollosi. Xgf - an xml game format. 2002. http://www.red-bean.com/sgf/xml/.

[Hollosi, 2006] Arno Hollosi. Sgf file format. 2006. http://www.red-bean.com/sgf/.

[Iida *et al.*, 2004] Hiroyuki Iida, Kazutoshi Takahara, Jun Nagashima, Yoichiro Kajihara, and Tsuyoshi Hashimoto. An application of game-refinement theory to mah jong. In Matthias Rauterberg, editor, *ICEC*, volume 3166 of *Lecture Notes in Computer Science*, pages 333–338. Springer, 2004.

[Ísleifsdóttir and Björnsson, 2008] Jónheidur Ísleifsdóttir and Yngvi Björnsson. Gtq: A language and tool for game-tree analysis. In *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*, pages 217–228. Springer, 2008.

[Khan *et al.*, 2008] Gul Muhammad Khan, Julian Francis Miller, and David M. Halliday. Coevolution of neuro-developmental programs that play checkers. In Gregory Hornby, Lukás Sekanina, and Pauline C. Haddow, editors, *ICES*, volume 5216 of *Lecture Notes in Computer Science*, pages 352–361. Springer, 2008.

[Kupferschmid and Helmert, 2006] Sebastian Kupferschmid and Malte Helmert. A skat player based on monte-carlo simulation. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. Donkers, editors, *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 2006.

[Lefler and Mallett, 2002] Mark Lefler and Jeff Mallett. Zillions of games, 2002. http://www.zillions-of-games.com.

[Love *et al.*, 2008] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General game playing: Game description language specification. 2008. http://games.stanford.edu/language/spec/gdl_spec_2008_03.pdf.

[Ontañón *et al.*, 2007] Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Case-based planning and execution for real-time strategy games. In Rosina Weber and Michael M. Richter, editors, *ICCBR*, volume 4626 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2007.

[Pell, 1994] Barney Pell. A strategic metagame player for general chesslike games. In *AAAI*, pages 1378–1385, 1994.

[Quenault and Cazenave, June 2007] Michel Quenault and Tristan Cazenave. Extended general gaming model. In *CGW 2007*, pages 195–204, June 2007.

[Santana *et al.*, 2004] Hugo Santana, Geber Ramalho, Vincent Corruble, and Bohdana Ratitch. Multi-agent patrolling with reinforcement learning. In *AAMAS*, pages 1122–1129. IEEE Computer Society, 2004.

[Schaeffer *et al.*, 2003] Jonathan Schaeffer, Martin Müller 0003, and Yngvi Björnsson, editors. *Computers and Games, Third International Conference, CG 2002, Edmonton, Canada, July 25-27, 2002, Revised Papers*, volume 2883 of *Lecture Notes in Computer Science*. Springer, 2003.

[Schaeffer *et al.*, 2007] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317:1518–1522, 2007.

[Schiffel and Thielscher, 2007] Stephan Schiffel and Michael Thielscher. Fluxplayer: A successful general game player. In *AAAI*, pages 1191–1196, 2007.

[Soeda *et al.*, 2006] Shunsuke Soeda, Tomoyuki Kaneko, and Tetsuro Tanaka. Dual lambda search and shogi endgames. In van den Herik et al. [2006], pages 126–139.

[van den Herik *et al.*, 2006] H. Jaap van den Herik, Shun chin Hsu, Tsan sheng Hsu, and H. H. L. M. Donkers, editors. *Advances in Computer Games, 11th International Conference, ACG 2005, Taipei, Taiwan, September 6-9, 2005. Revised Papers*, volume 4250 of *Lecture Notes in Computer Science*. Springer, 2006.

[van den Herik *et al.*, 2008] H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H. M. Winands, editors. *Computers and Games, 6th International Conference, CG 2008, Beijing, China, September 29 - October 1, 2008. Proceedings*, volume 5131 of *Lecture Notes in Computer Science*. Springer, 2008.