Gacrux software

Ver. 1.7, Otto Milvang, March 24, 2026
Ver. 1.8, Otto Milvang, May 11, 2026

Introduction

The **Gacrux software** is a collection of tools and shared data structures designed to support **Tournament Handler Programs (THPs)** for chess events. Its core purpose is to provide a consistent, transparent, and verifiable representation of tournament data—covering players, rounds, pairings, results, rankings, and tiebreaks—throughout the lifecycle of an event.

At the heart of Gacrux is a unified internal structure that can be **serialized to and from JSON**, referred to as the *Json Chess file (JCH)* format. This structure is designed to be human-readable, machine-processable, and easy to extend. By standardising how tournament data is stored and exchanged, Gacrux enables interoperability between different pairing engines, checking tools, web services, and external THPs.

Gacrux includes command-line programs for critical tournament operations such as **pairing generation, pairing verification, tiebreak calculation, ranking checks, and test tournament generation**. These tools can operate directly on JCH files or import and export common tournament formats, including **FIDE TRF-26**. This makes Gacrux suitable both as a standalone toolkit and as a backend engine embedded in larger tournament management systems.

Every operation produces structured output that includes status codes, diagnostics, and—when requested—detailed explanations of pairing and tiebreak decisions. This allows arbiters, developers, and organizers to verify correctness, compare alternative solutions, and trace decisions back to the applicable rules.

In short, Gacrux aims to be a **reliable reference implementation** for chess tournament processing, emphasizing correctness, reproducibility, transparency, and ease of integration.

JCH

The **Json Chess file (JCH)** defines the canonical data structure used by the Gacrux software to represent chess tournament information. It provides a structured, extensible, and machine-readable format for storing and exchanging all tournament-related data processed by Gacrux tools.

A JCH file encapsulates the complete state of a tournament, including identifiers, player and team data, rounds, pairings, results, rankings, and tiebreak information. The structure is designed to be serialized to and from JSON without loss of information, enabling deterministic processing and reliable interchange between different programs.

The JCH format is intentionally **implementation-neutral**. It does not prescribe how data is generated or modified, but instead defines a stable schema that can be consumed and produced by pairing engines, checking tools, web services, and external Tournament Handler Programs (THPs). This separation of data representation from algorithmic logic facilitates interoperability and long-term maintainability.

Each Gacrux program operating on a JCH file records its origin, options, status, and results in a consistent manner. As a result, every transformation of tournament data can be inspected, validated, and reproduced, supporting the overarching design goals of transparency and auditability.

The JCH definition is versioned. Backward-compatible extensions may be introduced as needed to support new rules, formats, or tournament types, while preserving the ability to process existing files.

Programs

Common program syntax

Command-line options are divided into two broad categories:
common options, which control input/output handling, encoding, verbosity, and tournament selection; and **program-specific options**, which define the operational behaviour of the individual tool. This separation allows users to apply the same execution pattern across different Gacrux programs while retaining fine-grained control over each operation.

`python programname.py [options]`

`-i, --input-file <filename>`

Path to input file.
"-" is used for standard input or piped input
Default: standard input

`-o, --output-file <filename>`

Path to output file.
"-" is used for standard output or piped output
Default: standard output

`-f, --input-format <format>`

Format of input file.
JCH – Json Chess file
TRF – FIDE Tournament Report FIle
TS – TournamentService files

Default: From file extension, and then JCH

`-F, --output-format <format>`

Format of output file.
JSON – Json program output
JCH – Json Chess file
TXT – Text output
TRF – FIDE Tournament Report File
TS – Not implemented

Default: If -d switch is present, then TXT, otherwise from file extension, and then JSON

`-b, --encoding <encoding>`

ascii, utf-8, cp1252 ...,
See https://docs.python.org/3/library/codecs.html#standard-encodings
for all possible values.
Default: Depends on file format. For TRF the default is ascii.

-e, --tournament-number <tournament number>

JCH and also some THPs support event-files with multiple tournaments, they are numbered 1 … number-of-tournaments. For some operations you want to apply the same operation on all tournaments, and then "0" means "all tournaments"

-n, --current-round < number>

See the actual meaning for each program

-N, --number-of-rounds < number>

The number of rounds in the tournament, overrides tournament value

-G, --game-score < list of key:val >
-M, --match-score < list of key:val >

You may define game score system from the command line. This may be useful for tournament generator, and other programs.

Kyes and values for –G and -M

Key	Record	Meaning	Default G	Default M
W	G/M	Win	1.0	2.0
D	G/M	Draw	0.5	1.0
L	G/M	Loss	0.0	0.0
Z	G/M	Zero point bye	0.0	0.0
P	G/M	Pairing allocated bye	D/W	D
A	G/M	Adjourned	D	D
F	G/M	Full point bye	W	W
H	G/M	Half point bye	D	D
U	G/M	Unknown	Z	Z
FG	M	Game points F		W*
HG	M	Game points H		D*
ZG	M	Game points Z		Z*
PG	M	Game points P		P*

Default Game points for P are W in individual tournaments, and D in team tournaments
X* (X=F,G,Z,P) is the game points for X times the number of players on the team.

Values must be numeric for W,D,L and Z,
for all other it may be letters or numeric

Example: set default PAB to draw for individual tournaments:
-G P:D

-c, --check

See the actual meaning for each program

-r, --rank

See the actual meaning for each program

-d, --delimiter <text>

This will force output format to be text, and for all output parameters are separated by the text
"T" – Tab
"B" – Blank
"S" – Semicolon
"@" – Print status code only (0 /1 / 2 / error code)
"C" – Comma
text – text

-D, --decimal-point <text>

This will use custom decimal point in text output
"C" – Comma
"P" – Point
text – text

-v, --verbose

Write additional information to stdout such as progress and timing
This is a count switch, so -v may give less information than -vv, and -vv may give less information than -vvv (and so on)

-V, --version

Add version number to stdout and exit

JSON output

A common JSON output has a header, status and content.

Mandatory fields are written as bold

Main object

Key	Type	Description
filetype	String	Defines the content of the file. It may be Event, Pairing, Tiebreak or other.
version	String	The version of this definition, on the form x.y or x.y.z. The major number x defines the version of this document.
Published	String	Date-time for creation
origin	String	Name of program
options	Object	All options
status	Object	Status object
<filetype>Result	Object	Result of operation. Examples: tiebreakResult, pairingResult, and so on

Options object

Key	Type	Description
<option 1>	Any	Value, List or default
...	...	...
<option N>	Any	Value, List or default

Status object

Key	Type	Description
code	Int	Status code, 0 is OK, if check option is on, then also check=True 1, is OK, if check option is on, then also check=False 2 is OK, but no legal operation (i.e. No legal pairing) 200, 201 and 202, same as 0,1 and 2, but used in chess web 400-499, error codes 500-599, internal errors
info	String array	A list of information records
error	String array	A list of error messages

Example:


```
{
  "filetype": "Pairing",
  "version": "1.0",
  "published": "2026-04-06 16:30:57",
  "origin": "pairingchecker ver. 1.7.50",
  "options": {
    "input_file": "test1315.trf",
    "output_file": "-",
    "input_format": "TRF",
    "output_format": "JSON",
    …,
    "tournament_number": "1",
    "current_round": 3
  }
  "status": {
    "code": 0,
    "error": []
  },
  "pairingResult":[ … ]
}
```


Pairingchecker.py

Synopsis

```
python pairingchecker.py [options]
```

Description

The **pairingchecker** program is a core analytical tool within the Gacrux software suite. Its primary purpose is to **generate, analyse, and verify pairings** in a chess tournament according to the applicable pairing rules.

In analyse mode, the program will read the tournament and find the pairing for the current round. For each score bracket analyse the quality criteria and exchange/transpose rules to find a vector representing the current pairing.

In pairing mode, the program will find the best pairing for the current round. For each score bracket optimize the pairing based on the quality criteria and exchange/transpose rules. Analyse this pairing to get a vector representing the new pairing.

In check mode the program will analyse the current pairing, then calculate its own pairing, and then compare the properties vector. If they are equal the pairing is correct.

The combinations of switches will determine how to present the result.

Options

In addition to -i, -o, -f, -F, -b, -e, -d, -v and -V the program accepts:

-m, --method <name>

The name of the pairing system.
The only legal value is “dutch” with variants “dutch-mp” and “dutch-gp” .
Default: dutch

The dutch algorithm will also work on team tournament (even if this is not supported by FIDE). The variants dutch-mp and dutch-gp will do pairing with respectively match points and game points as primary score.

-n, --current-round <round number>

The round to be paired or checked (the text “number of rounds” are misleading).
Default: For -p it’s the next round to be paired., for -a the last round paired, and -c all rounds.

-p, --pairing

Do pairing.

-a, --analysis

Do analysis.

-c, --check

Do check. Check if pairing is equal to analysis. Check flag without -p and -a flag will in text mode produce a summary for each round, and with -p and -a give a detailed analysis of the pairing process. In Json mode, -p and -a flag will give a lot of extra information.

-t, --top-color < w | b >

The color for the top player/team in round 1.
Defafault: random

-u, --unpaired < list of competitors >

A list of players/teams that shall not be paired in the next round

-K, --maxmeets < max number of meets >

Allow competitors to meet more than once. Experimental.
Default: 1

-T, --exchange < list of pairs >

Test exchange of pair. Used in combination with -c -a -p -n <round>. Suppose that the pairing contains 43-46, 47-58 and 44-75, however you think the correct are 44-58, 47-46 and 43-47, then you can simply test this with:

```
pairingchecker.py -i <inp> -a -p -c -F TXT -T 44-58 47-46 43-47 -n 3
```

The result output will give detailed information about why this pairing is better or worse.

-x, --experimental < list of argument >

"weighted" – Force algorithm to use weighted method for all players.
This will guarantee a correct pairing and used as a reference for optimization.
“time” – print elapsed time to stdout

Json output

The Json output is the recommended method.

The command line can be as simple as:

```
python pairingchecker.py -i <input> -o <output> -p
```

This will read a tournament in progress and return the pairing for the next round in the Json format described below.

```
python pairingchecker.py -i <input> -o <output> -c
```

Will check the pairing of paired round in a tournament. The “check” option force the program to produce quite detailed output.

The top node for Json output result is pairing Result that is just an array of objects.

pairingResult, pairing or analysis

For pairing or analysis the pairingResult is on the form

Key	Type	Description
Rules	String	Date of rules
Round	Int	Round to pair
Pairs	Array of pairs	Result of pairing, or analysis

Example

```
python pairingchecker.py -i ..\test\test1315.trf  -p -n 3
```

```
"pairingResult": [
    {
      "rules: "2026-02-01",
      "round": 3,
      "pairs": [1,6],[11,2],[5,10],[17,15],[7,3],[4,9],[12,16],[8,13],[14,0]]
    }
  ]
```

"pairs": [1,6],[11,2],[5,10],[17,15],[7,3],[4,9],[12,16],[8,13],[14,0]]

"pairs": [1,6],[11,2],[5,10],[17,15],[7,3],[4,9],[12,16],[8,13],[14,0]]

pairingResult, checker

If -c option is used, the program is in check-mode, and will test one or all rounds.

Key	Type	Description
rules	String	Date of rules
roundpairing	Array of objects	Array of detailed round info
check	Boolean	True if pairing == analysis

Example

```
python pairingchecker.py -i ..\test\test1315.trf  -c -n 3
```

```
"pairingResult": [
    {
      "rules: "2026-02-01",
      "roundpairing": [<array of roundpairing objects]
      "check": False
    }
  ]
```

roundpairing

Key	Type	Description
round	Int	result for round number <round>
pairs	Array of pairs	An array of pairs. This is the result from the pairingchecker.
current	Array of pairs	An array of pairs. This is the pairing in the tournament file.
check	Boolean	True if pairs == current reqires -c option.
pairing	Array of Objects	Pairingchecker pairing information.
analysis	Array of Objects	Tournament pairing information.
competitors	Array of Objects	Array of competitor objects
level2score	Array of floats	Translate scorelevel to scoreBracket points. In this definition PAB has score -1.0 points and is in scorelevel 0.

```
python pairingchecker.py -i ..\test\test1315.trf -c -p -n 3
```

```
"roundpairing": [
    {
      "round": 3,
      "pairs": [[1,6],[11,2],[5,10],[17,15],[7,3],[4,9],[12,16],[8,13],[14,0]],
      "current": [[1,6],[11,2],[5,10],[17,15],[7,3],[8,9],[4,16],[12,13],[14,0]],
      "check": false,
      "pairing": [{ <Pairing information }, … ],
      "analysis": [{ <Pairing information }, … ],
      "level2score": [ <float array > ]
    }
  ]
```

pairingInformation

Pairing information contains detailed information from each score bracket in the pairing process, or from analysis of existing tournaments.

Key	Type	Description
scorelevel	Int	Enumeration of scorebrackets, level2score translate scorelevel to score
competitors	Array of int	List of competitors

Pairs	Array of objects	Array of pair details
downfloater	Array of int	List of competitors
remaining	Array of int	List of competitors
Quality	Object of weights	This is the sum of pair quality. The best pairing has the lowest weight. Explain quality position in Pairing information, QC6 – QC21 and QN8 are quality criteria, HE1 – HE2 are parameters for heterogeneous pairing, HO1 – HO5 are parameters for homogeneous pairing.
Bsne	Array of int	BSN for homogeneous pairing
Bsno	Array of int	BSN for hetrogeneous pairing
Pab	Boolean	True if this if PAB level

Pair Details

For each pair there are a lot of information about the pairing. The exact content of the fields depends on the pairing method.

Pair details for dutch pairing:

Key	Type	Description
ca	Int	Competitor A
cb	Int	Competitor B, B>A
sa	Int	Scorelevel for A
sb	Int	Scorelevel for B
canmeet	Boolean	Can they meet
isblob	Boolean	Is B a blob (representing all players with "competitor-scorlevel" < "pairinginformation-scorelevel"- "pairinginformation-levels")
played	Int	Number of times A and B have met
unplayed	Int	Number of unplayed games by B (for PAB calculations)
qlevel	Int	Last level where quality was calculated
quality	Array of weights	Weight for one pair
weight	Int	Total weight
qcweight	Int	Weight C6-C21
heweight	Int	Weight heterogeneous pairing
howeight	Int	Weight homogeneous pairing
colordiff	Int	Last color played by A + Last color played by B
qc	String	True if 96-C11 and C12-C21 is 0
mode	Boolean	Find pairing mode
levels	String	how many levels was used, 0 if all
w	Int	Id for white player
B	Int	Id for blackplayer
colorrule	Int	Rule used for color assignment
board	String	Board number

Competitors

Competitors are an array of competitors. This contains detailed information used in the pairing algorithm.

Key	Type	Description
cid	Int	Competitor ID
rnk	Int	rank
pts	Float	Points
acc	Float	Points + Accellerated points
rfp	Boolean	Ready for pairing (present)
hst	Object	History
num	Int	Number of played games
rip	Int	Number of rounds paired (for TPN assignment)
cod	Int	Color difference
cop	String	Color preference (w0, w1, w2, b0, b1, b2)
csq	String	Color sequence
flt	Int	Downfloat/upfloat (8=df 4=uf in prev round, 2=df 1=uf in 2 rounds before)
top	Boolean	Top scorer
scorelevel	Int	Score level

level2score

Level2score is an enumeration of score. Pairing allocated bye will in this model has -1.0 point, and scorelevel 0. For all scores that exist in the tournament, the level will increase by 1.

```
"level2score": [
   "-1.0",
   "0.0",
   "1.0",
   "1.5",
   "2.0",
   "2.5",
   "3.0",
   "4.0"
 ]
```

In this example there are no players with 0.5 point or 3.5 points. In the enumeration scorelevel 0 => -1.0 point, scorelevel 1 = 0.0 points, scorelevel 2 = 1.0 points, …, scorelevel 7 => 4.0 points.

Text output

The Text output is easier to read for humans, and also compatible with previous pairing programs like JaVaFo and bbpPairings.

The command line can be as simple as:

```
python pairingchecker.py -i <input> -o <output> -p -dT
```

with response like:

```
9
3 6
7 1
2 13
5 9
16 15
17 4
12 14
8 10
11 0
```

The first line shows the number of pairs, in this example 9, and then the pairs. The last pair 11 0 means that 11 has Pairing allocated bye.

```
python pairingchecker.py -i <input> -o <output> -c -dT
```

Will check the pairing of paired round in a tournament. The "check" option will check all round played in a tournament.

```
================= Round #1 =================
================= Round #2 =================
================= Round #3 =================
 Checker pairing       Tournament analysis
    4 - 9              8 - 9
   12 - 16             4 - 16
    8 - 13            12 - 13
================= Round #4 =================
================= Round #5 =================
================= Round #6 =================
================= Round #7 =================
Check: False
```

Or simply:

```
python pairingchecker.py -i <input> -o <output> -c -d@
```

with response

0

If the tournament is correct, and 1 if its not correct.

python pairingchecker.py -i <input> -o <output> -c -a -p -dT -n

gives a deeper understanding of the pairing of round 3. For each scorebracket we got detailed information of the checker painting and the tournament analysis. This is an example of scorebracket 0.5 points. An arrow and will show the first (and only first) difference between analysis of existing pairing and pairing checker. It will also show where the pairing checker is better.

```
== Bracket: 0.5, scorelevel: 2
SNO BSN  PTS    P    A T CP CD F1 F2    1    2
  9   1 1.0   4b   8b   b0  0          1b  14w
  4   2 0.5   9w  16w   w1 -1         12b    Z
  8   3 0.5 down   9w   w1 -1         16b    -
 12   4 0.5  16w down   w0  0          4w   2b
 16   5 0.5  12b   4b   w0  0  D       8w   3b

Colors:
      w0:   2   b0:   1
      w1:   2   b1:   0
      w2:   0   b2:   0
Tot:  wc:   4   bc:   1   nc:   0

Pairs    :    Pairing      BSN        Analysis      BSN
              4 - 9    (   2 - 1   )   8 - 9    (   3 - 1   )  <=====  HE2
             12 - 16   (   4 - 5   )   4 - 16   (   2 - 5   )
Down     :   8 -      (   3 -     )  12 -      (   4 -     )

Rule   Checker pairing         Tournament analysis
QC6    1                      1
QC7    [0]                    [0]
QN8    1                      1
QC8    [0, 0]                 [0, 0]
QC9    0                      0
QC10   0                      0
QC11   0                      0
QC12   1                      1
QC13   0                      0
QC14   0                      0
QC15   0                      0
QC16   0                      0
QC17   0                      0
QC18   [0]                    [0]
QC19   [0]                    [0]
QC20   [0]                    [0]
QC21   [0]                    [0]
HE1    [0]                    [0]
HE2    [1]                    [2]
HO1    1                      0
HO2    1                      0
HO3    [0]                    [1]
HO4    [2, 3]                 [3, 3]
HO5    [0, 1]                 [2, 0]
```

tiebreakchecker.py

Synopsis

`python tiebreakchecker.py [options]`

Description

The **tiebreakchecker** program is responsible for the **calculation and verification of tiebreak values and ranking order** in a tournament. It applies the specified tiebreak rules to the tournament data and produces a structured description of the resulting scores and rankings. There are two modes, calculate mode and check mode.

In calculate mode, the program will calculate tiebreaks for the current round or after the final round. It will calculate values for all tiebreaks, and rank order for the competitors.

In check mode the program will calculate its tiebreaks and rank order of competitor, and then compare with the tournament. If they are equal the tiebreaks are correct.

The combinations of switches will determine how to present the result.

Options

In addition to -i, -o, -f, -F, -b, -e, -d, -v and -V the program accepts:

`-n, --current-round <round number>`

number of rounds that are used in the tiebreak calculation.
Default: all rounds.

`-t, --tiebreak <list>`

List of parameters used to determine the rank order, including points.
Default: PTS

`-p, --pre-determined`

Use rules for tournament with pre-determined pairing.

`-s, --swiss`

Use rules for swiss tournament.

`-u, --unrated < rating>`

Rating used in rating-based tiebreaks for unrated players.

Tiebreak list

Tie break specification

A Tiebreak is described with:

TB :pp /Mn /opt

- **TB** - name of TB, example: PTS (points), BH (Buchholz), DE (Direct encounter) ...
- **:pp** - Score modifier team tournaments, :MP = use match point, :GP = Use game points
- **/Mn** - Modifier in calculation, /C2 = cut 2, /M1 = median 1, /L+2 = Limit +2 1/2 points
- **/opt** - Options for calculations.

Tiebreaks

Point system					
Name	**Score**	**Modifiers**	**Section**	**Example**	**Comments**
PTS PTS MPTS GPTS MPvGP	:MP :GP		13.1	PTS PTS:GP MPTS GPTS MPvGP	Number of points in induvidual tournaments Number of points in team tournaments Number of points in team tournaments, Primary score is Match points Number of points in team tournaments, Primary score is Game points
Tiebreaks					
Name	**Score**	**Modifiers**	**Section**	**Example**	**Description**
DE		/P	6	DE	DirectEncounter
WIN	:MP		7.1		
WON	:MP		7.2		
BPG			7.3		
BWG			7.4		
PS	:MP :GP	/C1 /C2	7.5	PS	
REP			7.6	REP	Rounds elected to play, replaces GE, GE will still work
GE			7.6	REP	Depreciated, use REP (same functionality)
STD			7.7	STD	
BH	:MP :GP	/C1 /C2 /M1 /M2 /P	8.1	BH/C1	
AOB	:MP :GP	/F	8.2	AOB:GP/F	
FB	:MP :GP	/C1 /C2 /M1 /M2 /P	8.3	FB:MP/M2/P	
SB		/C1 /C2 /P	9.1	SB	SB:GP is identical to EGGSB, SB:MP is identical to EMMSB,
KS	:MP :GP	/L+n /L-n	9.2	KS /L-2	

ARO		/C1 /C2 /M1 /M2 /U1400	10.1	ARO/C1	
TPR		/U1400	10.2	TPR	
PTP		/U1400	10.3	PTP/U1400	
APRO		/U1400	10.4	APRO	
APPO		/U1400	10.5	APPO/U1200	
BC			12.1	BC	
TBR			12.2	TBR	
BBE			12.3	BBE	
ESB EMMSB EMGSB EGMSB EGGSB	:MM :MG :GM :GG	/C1 /C2 /P	13.2	ESB:MM EMMSB EMGSB/C1 EGMSB/P EGGSB/C2/P	 same as ESB:MM same as ESB:MG/C1 same as ESB:GM/P same as ESB:GG/C2/P
EDE		/P	13.3	EDE	
SSSC		/Kx /P /F	13.4	SSSC/K5	
Help functions					
Name	**Score**	**Modifiers**	**Section**	**Example**	**Comments**
SNO				SNO	Start number
RANK				RANK	Original rank in tournament file
RTG				RTG	Rating
RND				RND	Unique random number
VUR		/P	16.5	VUR	voluntary unplayed rounds
ABH		/P	16.3	ABH	Adjusted score for BH
AFB		/P	16.3	ABH	Adjusted score for FB
Functions to check pairing					
Name	**Score**	**Modifiers**	**Section**	**Example**	**Comments**

NUM				NUM	Number of played games
COP				COP	Color preference
COD				COD	Color difference
CSQ				CSQ	Color sequence
ACC				ACC	Points + accellerated points
FLT				FLT	Float (8=df 4=uf in prev round, 2=df 1=uf in 2 rounds before)
RFP				RFP	Registered for round
TOP				TOP	Is player a top-scorer in last round
Score selectors					
	Score		**Section**	**Example**	**Description**
				EDE	No selector is used for individual tournaments, or in team tournaments with primary score
	:MP		11	BH:MP	If primary score is GP, :MP will calculate Tie-break with MP instead of GP
	:GP		11	BH:GP	If primary score is MP, :GP will calculate Tie-break with GP instead of MP
	:MM			ESB:MM	Only valid for ESB
	:MG			ESB:MG	Only valid for ESB
	:GM			ESB:GM	Only valid for ESB
	:GG			ESB:GG	Only valid for ESB
Modifiers and options					
		Modifier		**Example**	**Description**
		/C1			Cut 1
		/C2			Cut 2
		/M1			Median 1
		/M2			Median 2

		/L+n /L-n		L+3	/L is modifier to KS, and tells how many half points the limit is moved up or down. n is integer
		/L+f /L-f		/L+1.0 /L-1.0	alternative syntax, where decimal number is the number of points to move the limit. /L+2 is equal to /L+1.0, f is decimal number
		/Ln		/L40	alternative syntax, where nuber is percent. In a five round tournament /L40 is equal to L-1 and /L-0.5
		/Kx		/K5	/K is modifier to SSSC, and is a scale factor
		/P		/P	Forfeit wins or losses are treated as regular games.
		/F		/F	Use Fore Buchholz insted of Buchholz. Note that FB is identical to BH/F
		/U1400		/U1400	Set unrated players to 1400 (or other value)
		/V1		/V1	Override start date, use 2024 rules
		/V2024		/V1	Override start date, use 2024 rules
		/V2		/V2	Override start date, use 2024 rules
		/V2026		/V6	Override start date, use 2026 rules
		/R		/R	Reverse order (useful for SNO, RTG, RAND, ...)

Json output

The Json output is the recommended method.

The command line can be as simple as:

`python tiebreakchecker.py -i <input> -o <output> -t <tiebreak list>`

This will read a tournament in progress and return the tiebreaks and rank for the current round in the Json format described below.

`python pairingchecker.py -i <input> -o <output> -c -t <tiebreak list>`

Will check the pairing of paired round in a tournament. The “check” option forces the program to produce quite detailed output

TiebreakResult

Key		Type	Description
rules		String	Date of rules
round		Int	result for round number <round>
tiebreaks		Array of Objects	Description of Tiebreaks
competitors		Array of Objects	Calculation of Tiebreaks
check		Boolean	check result if -c option

tiebreaks

Tiebreaks is an array of tiebreak objects.

Key	Type	Description
order	Int	Tiebreak number
name	String	Name of TB accoring to TB-list
pointtype	String	points, mpoints, or gpoints
modifiers	Object	List of options to TB
precision	Int	Number of decimals for presentation

Modifiers

An object of modifiers.

Key	Type	Description
ver	nuber	1 => rules before 2026-03-01, 2=> rules from 2026-03-01
rev	Boolean	False => Sort from low to high True => Sort from high to low
cutlow	Int	I.e BH, SB, PS
cuthigh	Int	Upper part of median cut
plim	Float	KS lim in persentage
nlim	Float	KS lim in score
unrated	Int	Set value to unrated players
predetrmined	Boolean	Treat unplayed games as played
swiss	Boolean	Treat as Swiss tournament
foremode	Boolean	Fore option
urd	Boolean	All unplayed are draw (experimental)

```
{
    "order": 2,
    "name": "BH",
    "pointtype": "gpoints",
    "modifiers": {
      "rev": true,
      "ver": 2,
      "cutlow": 1,
      "cuthigh": 0
    },
    "precision": 1
  }
```

competitors

Competitors is an array of competitor object with tiebreak-result for each competitor.

Key	Type	Description
cid	Int	Competitor id
rank	Int	Rank after computition
tiebreakScore	Array of float	The value of each tiebreak
tiebreakDetails	Array of Objects	Detailed information about the computition
boardCount	Array of Float	Score on each board in team tournaments

tiebreakDetails

An object with details about each round, and also the effect of modifiers.

Key	Type	Description
val	Float	The TB-value
<round>	Float	Contributions per round
cut	Array of Integers	Rounds to exclude

```
[
  {
    "cid": 1,
    "rank": 10,
    "tiebreakScore": [
      "2.0",
      "55.0"
    ]
    "tiebreakDetails": [

      {
        "val": "2.0",
        "1": "1.0",
        "2": "1.0",
        "3": "0.0",
        "4": "0.0",
        "5": "0.0",
        "6": "0.0"
      },
      {
        "val": "55.0",
        "cut": [
          5
        ],
        "1": "13.5",
        "2": "10.5",
        "3": "13.0",
        "4": "12.5",
        "5": "5.5",
        "6": "5.5"
      }
    "boardPoints": {
      "1": "1.5",
      "2": "0.5",
      "3": "2.5",
      "4": "1.0"
    }
  }, …
]
```

Text output

The command line for text output:

python tiebreakchecker.py -i <input> -o <output> -t <tiebreak list> -d T

This will read a tournament in progress and return the tiebreaks and rank for the current round in formatted text described below.

python pairingchecker.py -i <input> -o <output> -c -t <tiebreak list> -d T

This will give the same output with one additional line:
Check: True/False

python tiebreakchecker.py -i test1315.trf -t PTS DE BH BH/C1 WON -d T

```
StartNo Rank    PTS     DE      BH      BH/C1   WON
1       1       5.0     1       29.5    26.0    4
2       2       5.0     2       25.5    23.5    2
3       4       4.5     0       28.0    25.0    4
4       15      2.5     0       21.5    19.0    1
…
```

Note that T in -dT is TAB, other predefined variables is B-blank C-comma, txt for tree text, and @ to only print the status code

```
D:\Dropbox\Privat\spp\TieBreakChecker\TieBreakChecker>python
tiebreakchecker.py -i ..\test\test1315.trf -t PTS DE BH BH/C1 WON -dS
StartNo;Rank;PTS;DE;BH;BH/C1;WON
1;1;5.0;1;29.5;26.0;4
2;2;5.0;2;25.5;23.5;2
3;4;4.5;0;28.0;25.0;4
4;15;2.5;0;21.5;19.0;1
…
```

```
D:\Dropbox\Privat\spp\TieBreakChecker\TieBreakChecker>python
tiebreakchecker.py -i ..\test\test1315.trf -t PTS DE BH BH/C1 WON -c -d@
1
```

(0 is OK, 1 is check failed)

tournamentgenerator.py

Synopsis

`python tournamentgenerator.py [options]`

Description

T he **tournamentgenerator** program is a supporting tool intended primarily for **testing, validation, and development** rather than for operational tournament management. Its purpose is to generate large numbers of synthetically constructed chess tournaments with controlled statistical properties, using the same pairing engine and rule interpretations as the Gacrux pairing checker.

The generated tournaments are exported in **TRF-26** format and are suitable as direct input to other Gacrux tools, including pairingchecker and tiebreakchecker. This ensures that all generated data can be processed, analysed, and verified under identical conditions to real tournaments.

The tournamentgenerator uses the **same pairing logic** as the pairing checker. As a result, all generated pairings conform to the same system constraints and rule interpretations that are applied to real tournaments. This alignment is intentional and ensures that test data is representative of realistic tournament behaviour.

Player ratings, results, byes, forfeits, and optional acceleration are generated according to parameterized statistical models. These parameters allow controlled variation while preserving internal consistency within each generated tournament.

The program is deterministic with respect to its input parameters, except where randomness is explicitly introduced. This makes it suitable both for exploratory testing and for reproducible test scenarios.

Options

In addition to --N, -v and -V the program accepts:

`-g, --generate [number-of-tournaments]`
`-g, --generate [startno number-of-tournaments]`
`-g, --generate [startno number-of-tournaments step]`

number of tournaments to be generated, a range of tournament, and also range with increment.
Default: 1000.

`-o, --output-file`

%d in the name will be replaced by 4 digits 0000 to 9999.

`-p, --players <number of competitors>`

number of competitors in the tournament.
Default: 40.

`-m, --method <name on pairing method>`

Only dutch is implemented.
Default: dutch.

`-t, --top-color <w | b>`

Color for TPN 1 in round 1.

-T, --members <team size>

Number of players per team (playing).

-R, --rating [top step sigma]

First parameter is rating strength for top rated player.
Second parameter is step in rating strength between players (float)
Third parameter is sigma, a statistical variance such that player n will have rating
rating = round(top – step *(n-1) + N(0,sigma)
where N is the normal distribution with mean 0, and st. dev sigma.
Default values depends on number of players.
< 65: top = 2200, step = 10, sigma = 50.0

-S, --statistics [zpb hpb forfeited]

The rate of zero point bye, half point bye and forfeited games
The defaults are zpb=0.01, hpb=0.05 (only 3 round, then 0.0), forfeited=0.02

-a, --acceleration

Use Baku acceleration.

Example:

python tournamentgenerator.py -g 500 -n 9 -p 300 -o c:\files\myfilename%d.trf

Generates 500 tournaments

chessserver.py

Synopsis

```
python chessserver.py
```

Description

This is an api-service for web. If the webserver is configured to run Python, this is a the entryp point for processing data. Its implemented 3 services:

- convert, convert a file to JCH format.
- pairing, runs pairingchecker
- tiebreak, runs tiebreakchecker

chessserver will take input from <stdin> and simply run one of the services, and then write the result to <stdout>

Request

A JCH formatted message is sent to the web-server as a POST request

Request:

```
{
    "filetype": "convert request" | "tiebreak request" ,
    "version": "1.0",
    "origin": "<Free text>",
    "published": "<date on format 2018-08-14 05:07:44>",
    "options": {
        "service" : "convert" | "pairing" | "tiebreak",
        "base64": ["<lines with base 64 encoded file>"],
        "input_filename" : "<original file name>",
        "input_filetype": "TRF" | "TS" | < other known format >,
        …,
        // parameters for tiebreaks
            "tiebreak" : [string list],
            "pre_determined" : true | false,
            …,
        // parameters for pairing
            "pairing" : true | false,
            "method" : "dutch",
            "top_color" : "white" | "black",
            …,
    }
}
```

Key	Type	Description
service	String	Then name of the service
Base64	String	A base-64 coded tournament file
<other options>	String	Same as options for programs with - replaced by _ Example current-round => current_round

Response

A JCH formatted message identical to the Json output from the service.