GNU Backgammon

version 0.14.2-devel
October, 2003

Gary Wong, Achim Mueller

1 Introduction

GNU Backgammon works on

- MS Windows operating systems
- Linux
- FreeBSD
- Solaris
- MacOS 10

GNU Backgammon is available in different languages

- English
- German (nearly finished)
- Italian (3/4 finished)
- Japanese (2/3 finished)
- Danish (1/4 finished)
- Turkish (1/4 finished)
- Icelandic (1/4 finished)
- French (not ready yet)

GNU Backgammon comes with

- A Graphical User Interface (GUI) with 2D- or 3D-Animation
- Including a selection of board designs to choose from
- Powerful but simple tools to change or create your own design
- You can choose colours, lighting, textures and even the perspective of the 3D board
- A command line modus (CLI) with support of the script language python

With GNU Backgammon you may play

- backgammon, nackgammon or hypergammon
- games, matches or sessions (`gnubg` adjusts its play always to the match score)

Gnubgs analysing functions allow you to

- get hints during the game
- evaluate positions
- use a tutor during the game
- use a "temperature map" which gives a graphical simple overview of the possible results of the next dice roll
- analyse or rollout positions, games, matches or sessions (with probably the most extensive controls to set speed and accuracy of any backgammon software)

You can

- setup, edit and save positions, games, matches and sessions
- replay games, matches and sessions
- record annotations
- explore doubling windows, gammon prices and similar cube theory, valuable for money games and essential for match play
- store error rates for different players

GNU Backgammon imports matches and sessions from

- Fibs
- Gamesgrid
- TrueMoneygame
- Netgammon
- Snowie
- Jellyfish

GNU Backgammon exports positions, games, matches and sessions into

- *.pos
- *.mat
- plain text
- different html formats
- png
- LaTeX
- (Encapsulated) Postscript
- PDF

With GNU Backgammon you may choose between the following dice generators

- Manual dice settings
- from a file
- a dice generator created by the user
- Ansi
- Blum, Blum and Shub
- BSD
- ISAAC
- MD5
- Mersenne Twister
- www.random.org

GNU Backgammon comes with several databases for handling bearoffs and match equities, i.e.

- larger bearoff databases (to be generated)
- databases for 1/2/3-chequer hypergammon (to be generated)
- a selection of several Match Equity Tables (METs)
- an interface to Hugh Sconyers huge databases for races and bearoffs

But don't forget ... GNU Backgammon is work in progress. Don't blame us if

- a feature isn't implemented yet
- a feature doesn't work correctly
- `gnubg` crashes
- your system suffers after gnubg is installed

Don't forget also ...

- it's free software (free as in "free speech", not in "free beer")
- we are all volunteers
- we need feedback and bug reports
- we need your help

2 For the impatient

2.1 Installing the binaries

2.2 Playing a game

3 How to play backgammon

The rules presented in this chapter were written by Tom Keith for the Backgammon Galore web site (`http://www.bkgm.com/`), and are included here with his permission.

3.1 Rules of Backgammon

3.1.1 Setup

Backgammon is a game for two players, played on a board consisting of twenty-four narrow triangles called *points*. The triangles alternate in color and are grouped into four quadrants of six triangles each. The quadrants are referred to as a player's *home board* and *outer board*, and the opponent's home board and outer board. The home and outer boards are separated from each other by a ridge down the center of the board called the *bar*.

Figure 1. checkers i
tion.

An altern
the revers
here, with
on the lef
board on

The points are numbered for either player starting in that player's home board. The outermost point is the twenty-four point, which is also the opponent's one point. Each player has fifteen checkers of his own color. The initial arrangement of checkers is: two on each player's twenty-four point, five on each player's thirteen point, three on each player's eight point, and five on each player's six point.

Both players have their own pair of dice and a dice cup used for shaking. A *doubling cube*, with the numerals 2, 4, 8, 16, 32, and 64 on its faces, is used to keep track of the current stake of the game.

3.1.2 Object of the Game

The object of the game is for a player to move all of his checkers into his own home board and then bear them off. The first player to bear off all of his checkers wins the game.

Figure 2.
ment of W
Red's chec
opposite d

3.1.3 Movement of the Checkers

To start the game, each player throws a single die. This determines both the player to go first and the numbers to be played. If equal numbers come up, then both players roll again until they roll different numbers. The player throwing the higher number now moves his checkers according to the numbers showing on both dice. After the first roll, the players throw two dice and alternate turns.

The roll of the dice indicates how many points, or *pips*, the player is to move his checkers. The checkers are always moved forward, to a lower-numbered point. The following rules apply:

1. A checker may be moved only to an *open point*, one that is not occupied by two or more opposing checkers.

2. The numbers on the two dice constitute separate moves. For example, if a player rolls 5 and 3, he may move one checker five spaces to an open point and another checker three spaces to an open point, or he may move the one checker a total of eight spaces to an open point, but only if the intermediate point (either three or five spaces from the starting point) is also open.

Figur
Whit
53.

3. A player who rolls doubles plays the numbers shown on the dice twice. A roll of 6 and 6 means that the player has four sixes to use, and he may move any combination of checkers he feels appropriate to complete this requirement.

4. A player must use both numbers of a roll if this is legally possible (or all four numbers of a double). When only one number can be played, the player must play that number. Or if either number can be played but not both, the player must play the larger one. When neither number can be used, the player loses his turn. In the case of doubles, when all four numbers cannot be played, the player must play as many numbers as he can.

3.1.4 Hitting and Entering

A point occupied by a single checker of either color is called a *blot*. If an opposing checker lands on a blot, the blot is *hit* and placed on the bar.

Any time a player has one or more checkers on the bar, his first obligation is to *enter* those checker(s) into the opposing home board. A checker is entered by moving it to an open point corresponding to one of the numbers on the rolled dice.

For example, if a player rolls 4 and 6, he may enter a checker onto either the opponent's four point or six point, so long as the prospective point is not occupied by two or more of the opponent's checkers.

Figure 4. with a che
must ente
Red's fou
six point i

If neither of the points is open, the player loses his turn. If a player is able to enter some but not all of his checkers, he must enter as many as he can and then forfeit the remainder of his turn.

After the last of a player's checkers has been entered, any unused numbers on the dice must be played, by moving either the checker that was entered or a different checker.

3.1.5 Bearing Off

Once a player has moved all of his fifteen checkers into his home board, he may commence *bearing off*. A player bears off a checker by rolling a number that corresponds to the point on which the checker resides, and then removing that checker from the board. Thus, rolling a 6 permits the player to remove a checker from the six point.

If there is no checker on the point indicated by the roll, the player must make a legal move using a checker on a higher-numbered point. If there are no checkers on higher-numbered points, the player is permitted (and required) to remove a checker from the highest point on which one of his checkers resides. A player is under no obligation to bear off if he can make an otherwise legal move.

Figure 5.
bears off t

A player must have all of his active checkers in his home board in order to bear off. If a checker is hit during the bear-off process, the player must bring that checker back to his home board before continuing to bear off. The first player to bear off all fifteen checkers wins the game.

3.1.6 Doubling

Backgammon is played for an agreed stake per point. Each game starts at one point. During the course of the game, a player who feels he has a sufficient advantage may propose doubling the stakes. He may do this only at the start of his own turn and before he has rolled the dice.

A player who is offered a double may *refuse*, in which case he concedes the game and pays one point. Otherwise, he must *accept* the double and play on for the new higher stakes. A player who accepts a double becomes the *owner of the cube* and only he may make the next double.

Subsequent doubles in the same game are called *redoubles*. If a player refuses a redouble, he must pay the number of points that were at stake prior to the redouble. Otherwise, he becomes the new owner of the cube and the game continues at twice the previous stakes. There is no limit to the number of redoubles in a game.

3.1.7 Gammons and Backgammons

At the end of the game, if the losing player has borne off at least one checker, he loses only the value showing on the doubling cube (one point, if there have been no doubles). However, if the loser has *not* borne off any of his checkers, he is *gammoned* and loses *twice* the value of the doubling cube. Or, worse, if the loser has not borne off any of his checkers and still has a checker on the bar or in the winner's home board, he is *backgammoned* and loses *three times* the value of the doubling cube.

3.1.8 Optional Rules

The following optional rules are in widespread use.

Automatic doubles
: If identical numbers are thrown on the first roll, the stakes are doubled. The doubling cube is turned to 2 and remains in the middle. Players usually agree to limit the number of automatic doubles to one per game.

Beavers
: When a player is doubled, he may immediately redouble (beaver) while retaining possession of the cube. The original doubler has the option of accepting or refusing as with a normal double.

The Jacoby Rule
: Gammons and backgammons count only as a single game if neither player has offered a double during the course of the game. This rule speeds up play by eliminating situations where a player avoids doubling so he can play on for a gammon.

3.1.9 Irregularities

1. The dice must be rolled together and land flat on the surface of the right-hand section of the board. The player must reroll both dice if a die lands outside the right-hand board, or lands on a checker, or does not land flat.
2. A turn is completed when the player picks up his dice. If the play is incomplete or otherwise illegal, the opponent has the option of accepting the play as made or of requiring the player to make a legal play. A play is deemed to have been accepted as made when the opponent rolls his dice or offers a double to start his own turn.
3. If a player rolls before his opponent has completed his turn by picking up the dice, the player's roll is voided. This rule is generally waived any time a play is forced or when there is no further contact between the opposing forces.

3.2 Rules for Match Play

When backgammon tournaments are held to determine an overall winner, the usual style of competition is *match play*. Competitors are paired off, and each pair plays a series of games to decide which player progresses to the next round of the tournament. This series of games is called a *match*.

Matches are played to a specified number of points. The first player to accumulate the required points wins the match. Points are awarded in the usual manner: one for a single game, two for a gammon, and three for a backgammon. The doubling cube is used, so the winner receives the value of the game multiplied by the final value of the doubling cube.

Matches are normally played using the *Crawford rule*. The Crawford rule states that if one player reaches a score one point short of the match, neither player may offer a double in the immediately following game. This one game without doubling is called the Crawford game. Once the Crawford game has been played, if the match has not yet been decided, the doubling cube is active again.

Match to 5		**White**	**Black**	
Game 1:	White wins 2 points	2	0	Doubling Allowed
Game 2:	Black wins 1 point	2	1	
Game 3:	White wins 2 points	4	1	
Game 4:	Black wins 1 point	4	2	Crawford Game
Game 5:	Black wins 2 points	4	4	Doubling Allowed
Game 6:	White wins 2 points	6	4	

In this example, White and Black are playing a 5-point match. After three games White has 4 points, which is just one point short of what he needs. That triggers the Crawford rule which says there can be no doubling in next game, Game 4.

There is no bonus for winning more than the required number of points in match play. The sole goal is to win the match, and the size of the victory doesn't matter.

Automatic doubles, beavers, and the Jacoby rule are not used in match play.

4 Installing the software

4.1 Installing a MS Windows binary

For an installation of GNU Backgammon on MS Windows operating systems you need at least this build from gnubg. You'll find a file '`setup.exe`', which is approximately 11 mb. If you want daily builds, go to Nardy Pillards page at `http://users.skynet.be/bk228456/GNUBgW.htm` and read his documentation carefully. If you download everything, you'll have these packages:

- setup.exe (Oysteins installation archiv from 20030727, including everything you need)
- gnubg-old-[date]Zip.exe (Nardy's daily snapshot, no 3d-board, old layout)
- gnubg-[date]Zip.exe (Nardy's daily snapshot, with 3d-baord and new layout)
- board-3Dfiles.exe (Package needed for 3D-support)
- locale_[lang].exe (Nardy's language packages)
- gnubg_nogui-[date]Zip.exe (Nardy's daily snapshot without Graphical User Interface)
- libpython22Zip.exe (Package needed for python support in the "no gui" version)

Nardy's builds include a daily snapshot of the '`gnubg.exe`' and some addons. If you like daily builds, get either the "old layout", the "new layout" or the "no gui" binary (*.exe). You may also download a language package for Windows. Be aware, that the translation is still work in progress. Only the english and the german translation are (nearly) fully available.

Part I (Main program): After downloading the software "doubleclick" on '`setup.exe`'. You will be asked whether you want to install GNU Backgammon for windows. Click on *yes*. In the following window accept the license agreement and click on *next*. Then choose a directory where GNU Backgammon shall be installed. After this select the MS Windows menu entry and whether or not a desktop and a quick launch icon shall be created.

The installation program will install all necessary files on your disk and finally ask you whether GNU Backgammon shall be launched or not. Now you are ready to play.

Part II (Daily snapshots): If you like current snapshots, get the files from Nardy's page at `http://users.skynet.be/bk228456/GNUBgW.htm`. The files '`gnubg-[date]Zip.exe`' or '`gnubg-old-[date]Zip.exe`' contain only the binary '`gnubg.exe`'.

If you like the new style with panels and a 3D-board layout get '`gnubg-[date]Zip.exe`' and the additional package '`board-3Dfiles.exe`'. Doubleclick on the file, select your installation directory and click on *extract*. Be catious: the binary will now overwrite the "old" '`gnubg.exe`'.

For the old layout (seperate windows, no 3D-board) choose '`gnubg-old-[date]Zip.exe`'. You will now have a second program file in you GNU Backgammon directory called '`gnubg-old.exe`'.

Part III (Support for different languages: If there is a package supporting your native language, you may download this package also. Doubleclick on the '`locale_[lang].exe`', and the installer will put all necessary files into your GNU Backgammon directory. After copying the files a README file will open with instructions how to activate the language.

Part IV (Command Line Interface): For using the "non-gui" version of GNU Backgammon you only have to additionally install '`gnubg_nogui-[date]Zip.exe`'. Again doubleclick on the binary '`gnubg_nogui-[date]Zip.exe`'. Be aware that possible older version of '`gnubg-no-gui.exe`' will be overwritten. Accept the warning and click on *ok*. Choose the directory of your GNU Backgammon installation and click on *extract*.

Part V (Additional python support): If you need additional support for the scripting language python you also have to install the package '`libpython22Zip.exe`'.

4.2 Installing a Linux rpm-package

Get the rpm packages from the Linux download section at GNU Backgammon webpage. You'll need the main package '`gnubg-[version].rpm`', the databases '`gnubg-databases.rpm`' and the sound packages '`gnubg-sound.rpm`'. As root do a

```
#: rpm -ihv gnubg*.rpm
```

If you want to update GNU Backgammon you will mostly need only the new main package. Then you can update `gnubg` with

```
#: rpm -Uhv gnubg-[version].rpm
```

Some people like to compile GNU Backgammon from the source-rpms. In this case you have to download the package '`gnubg-[version].src.rpm`'. As root do a

```
#: rpm --rebuild gnubg-[version].src.rpm
```

For people who want to verify the packages we put a public key on public key ace. Download the key and import it with

```
#: gpg --import pubkey_ace.asc
#: rpm --verify [package] (for rpm version 3)

or

#: rpm --import pubkey_ace.asc
#: rpm --checksig [package] (for rpm version 4)
```

4.3 Installing a FreeBSD package or port

fixme

4.4 Compiling and installing the source

Download the sources from GNU Backgammon sources (`http://www.acepoint.de/GnuBG/sources/`). We try to put daily snapshots there. Be aware, that these sources may not compile!

Unzip and untar the sources and change into the source directory:

```
#: tar -xzvf gnubg-[version].tar.gz
#: cd gnubg-[version]
```

Copy at least the file '`gnubg.weights`' into the source directory. Then do a

```
#: aclocal -I m4
#: autoconf
#: automake --add-missing
```

```
#: ./configure
#: make
#: make install
```

The first three commands may not be necessary depending on your system. The command `./configure` includes some options you may like to use. You can get all options with

```
#: ./configure --help
```

5 Starting GNU Backgammon

5.1 The graphical interface

5.2 Using a terminal

6 Playing backgammon

6.1 Starting a game or match or session

6.2 Moving the checkers

6.3 Using the doubling cube

7 Saving games

7.1 Saving a game

7.2 Saving a match

7.3 Saving a session

7.4 Saving a position

8 Opening files

8.1 Opening a game

8.2 Opening a match

8.3 Opening a session

8.4 Opening a position

9 Import files

`gnubg` can import matches produced by various other programs or online servers. For example, you can import matches played on GamesGrid (`http://www.gamesgrid.com/`) or FIBS (`http://www.fibs.com/`). `gnubg`can export positions, matches, or money session to a number of formats; some suitable for printing, some suitable for including in emails or Usenet postings, and others suitable for publishing on the world wide web or for posting to backgammon forums.

The following sections describe how to import and export positions, matches, and sessions with `gnubg`.

9.0.1 Jellyfish .pos

`gnubg` can import positions in Jellyfish (`http://jelly.effect.no/`) .pos format. To import a position file using the GUI select a file from `File->Import->.pos position`.

The .pos position format is documented at The Jellyfish Position File Format (`http://jelly.effect.no/fspec.htm`).

9.0.2 Jellyfish .mat

`gnubg` can also import matches and money sessions in the .mat format. This format is often used to move games between different programs, as most programs (e.g., Jellyfish, Snowie, BGBlitz, and others) support import and export of this format.

To import a match or money session in the .mat format, select a file from `File->Import->.mat match`.

Note that at least one program (BBGT) is known to write multiple matches in the same .mat file. In this case `gnubg` will only read the first match. If you wish to import all matches from such a .mat file, you'll manually have to split the file into files containing the individual matches.

9.0.3 GamesGrid .sgg

`gnubg` can import matches and money sessions played on GamesGrid if saved in the SGG format. Currently, `gnubg` doesn't support import of the native undocumented .cbg format. Also note that `gnubg` does not analyse table stakes, so imported money sessions will be analysed as a normal money session.

To import an SGG file using the GUI select a file using the menu entry `File->Import->.sgg match`.

9.0.4 FIBS oldmoves

`gnubg` can import matches and money sessions from FIBS in the oldmoves format. Most of the GUIs for playing on FIBS supports saving of matches either in the oldmoves format or in Jellyfish .mat format.

To import an oldmoves file using the GUI select a file using the menu entry `File->Import->FIBS oldmoves`.

Note that `gnubg` will only read the first match in the file. If you wish to import all matches from a file containing more than more game, you'll manually have to split it.

9.0.5 TrueMoneyGames .tmg

`gnubg` can also import matches and money sessions played on TrueMoneyGames saved in the TMG format (file suffix .tmg). Please note that `gnubg` does not analyse table stakes, so imported money sessions will be analysed as a normal money session.

To import an TMG file using the GUI select a file using the menu entry `File->Import->TrueMoneyGames .tmg match`.

9.0.6 Hans Berliner BKG format

Matches written in the Hans Berliner BKG format can be imported into `gnubg`. To import an BKG file using the GUI select a file using the menu entry `File->Import->BKG session`.

9.0.7 Snowie .txt position format

`gnubg` can import positions written in the Snowie .txt position format. Please note that very important distinction between *Snowie .txt position* and *Snowie Standard Text*. The former is used for position and the latter is an extension of the Jellyfish .mat format, and can be imported in `gnubg` using the .mat importer.

`gnubg` can also export positions to Snowie .txt format for easy exchange of positions with Snowie users.

To import an Snowie .txt position file using the GUI select a file using the menu entry `File->Import->Snowie .txt position file`.

9.0.8 Batch import

`gnubg` doesn't support batch import and analysis using the GUI, but batch import is possible with the command line version of `gnubg`.

Under Unix execute:

`gnubg -t <` *input-file* `>` *output-file*

Under Windows it depends on how your version of gnubg was build. For example, if you use the build by ystein Johansen you should execute

`gnubg-no-gui.exe <` *input-file* `>` *output-file*

The *input-file* must be carefully prepared. For example, it could look like:

```
import sgg file1.sgg
analyse match
save match file1.sgf
import sgg file2.sgg
analyse match
save match file1.sgf
import mat file3.mat
analyse match
save match file3.sgf
```

If you wish to adjust your setting for the analysis you can include commands like:

```
set analysis chequerplay evaluation plies 2
set analysis cubedecision evaluation plies 3
[...]
```

See the command reference for further details regarding the commands available in `gnubg`.

9.0.9 Other formats

There are a number of utilities written that can read and write positions, matches, and money sessions in other formats than the ones `gnubg` has direct support for.

Bunny's Zone Converter (`http://www.wingflyer.com/zoneutilities/`) can be used to convert matches saved on MSN Gaming Zone to Jellyfish .mat format which can be imported into `gnubg`.

The neural net backgammon program BGBlitz (`http://www.bgblitz.com/`) can read Gamesgrid .cbg, FIBS/W logfiles, MacFIBS logfiles and other formats which can be written back in the .mat format for importing into `gnubg`.

9.0.10 Tips and tricks

If you want to exchange the playes, i.e., `gnubg` has imported you as the top player rather than the bottom player, select them menu entry `Game->Swap Players`.

10 Export files

10.0.1 Export game in Jellyfish .gam

You can export the current game in the Jellyfish .gam format. This is useful if you wish import a specific game into Jellyfish. Other programs may read the file as well, since the format is identical to the Jellyfish .mat format.

export game gam *filename*: Export the current game in Jellyfish .gam format to *filename*.

10.0.2 Export positions, games, matches and sessions in HTML

`gnubg` can export the current position, game, match or session in HTML if you wish to publish it on the web. A sample match (`http://home.online.no/~oeysteij/wc2002/final.html`) with the World Cup Final 2002 from Monte Carlo is available for viewing (FIXME: put a match on gnu.org or gnubg.org instead).

`gnubg` exports in validating XHTML 1.0 (`http://www.w3.org/TR/xhtml1/`) with the use of CSS (`http://www.w3.org/TR/REC-CSS1`) style sheets. You may add your own style sheet to the exported HTML files if you wish to override the default layout, e.g., change colors or fonts.

The board is made up from hundreds of pictures. Currently, you can choose between three different sets of pictures: (a) the BBS images used by Kit Woolsey's GammOnLine (`http://www.GammOnLine.com`) e-magazine, Danish Backgammon Federation's (`http://www.dbgf.dk/`) web-based discussion group and others, (b) the fibs2html images used by the Joseph Heled's program fibs2html (`http://fibs2html.sourceforge.net/`), and (c) images generated by `gnubg` itself. The images generated by `gnubg` will have the same design as your current design, and honours your settings on clockwise or anti-clockwise movement and board numbering (on, off, dynamic).

If you export a match or session to HTML `gnubg` will write the individual games to separate files. For example, if you export to file '`foo.html`' the first game is exported to '`foo.html`', the second game to '`foo_002.html`', the third game to '`foo_003.html`' and so forth.

The output from the HTML export can be customised. For example, it's possible to leave out the analysis or parts of the analysis. Also, you may enter a specific URL to the pictures used to compose the board which is useful for posting positions on web-based discussion groups such as Kit Woolsey's GammOnLine (`http://www.GammOnLine.com`), the Danish Backgammon Federation's (`http://www.dbgf.dk/`) Debat Forum (`http://www.dbgf.dk/Debat/`), or you may opt to use a default set of images available from the `gnubg` web site at FIXME Add images to www.gnubg.org! (`http://www.gnubg.org/html-images/`).

See ⟨undefined⟩ [Export], page ⟨undefined⟩, for full details regarding the options available for HTML export.

10.0.2.1 Description of the CSS stylesheet

As mentioned above `gnubg` writes a CSS stylesheet along with the generated XHTML file. The CSS stylesheet may be written verbatim in the header section of the XHTML file, to an external file named '`gnubg.css`', or inside the tags using the '`style`' attribute. If you wish to make any modifications to the stylesheet without modifying the actual source code of `gnubg` you have to choose one of the first two methods. Note that the special export for Kit Woolsey's GammOnLine (`http://www.GammOnLine.com`) uses the third method since the XHTML is pasted into a web page without the possibility to modify the header section of the page where the style sheet is defined. Thus, it's not possible to modify the style of the generated XHTML for GammOnLine without modifications of the source code or extensive *search and replace* in the geneated XHTML.

Below follows a description of the CSS classes used in the XHTML export:

Class	Description
.movetable	Style applied to the entire table used for the move analysis
.moveheader	The header of the move analysis table
.movenumber	The rank number of a move in the move analysis
.moveply	The column indicating the number of plies or rollout
.movemove	The formatted move, e.g., 13/7 8/7.
.moveequity	The column with the equity or MWC.
.movethemove	Special style for row that contains the actual move chosen by the player
.moveodd	Special style for the odd rows. Can be used to give an alternating color for the rows.
.percent	Style for the game winning probabilities and equities in the move analysis.
.blunder	Emphasize blunders, e.g., "Alert: missed double" or "Alert: bad move".
.joker	Emphasize very good or bad rolls, e.g., "Alert: unlucky roll".
.stattable	The style applied to the entire table with game, match, and session statistics
.stattableheader	The header row of the statistics table
.result	Style for the text indicating the outcome of the game or match, e.g., "Joern Thyssen wins 16 points".
.tiny	Currently unsued.
.cubedecision	The style applied to the entire cube decision table
.cubedecisionheader	Style for the header row of the cube decision table
.cubeequity	Style for any equity or MWC in the cube decision table
.cubeaction	Style for the text indicating the correct cube action
.cubeply	Style for the text that states the level of evaluation
.cubeprobs	Style for the game winning probabilities in the cube decision table

.comment	The style applied to the entire table used for annotations or comments, e.g., the kibitzing from imported SGG files
.commentheader	The style applied to the header row of the annotations' table
.number	Currently unsued
.fontfamily	Style applied to the entire body of the XHTML document.
.block	Style applied to the images in the export to avoid gaps between individual pictures both horisontally and vertically.
.positionid	Style for the Position ID and match ID.

export position html *filename*: Export the current position in HTML to *filename*.

export game html *filename*: Export the current game in HTML to *filename*.

export match html *filename*: Export the current match in HTML to *filename*.

export session html *filename*: Export the current session in HTML to *filename*.

10.0.3 Export games, matches, and sessions in LaTeX

writeme

10.0.4 Export match or session in Jellyfish .mat

You can export an entire match or session into Jellyfish .mat format. This is very useful if you want to import a match or session into other programs as most other backgammon programs are able to read it.

export match mat *filename*: export the match in Jellyfish .mat format to *filename*.

10.0.5 Export games, matches, and sessions in PDF

writeme

10.0.6 Export games, matches, and sessions in PostScript

writeme

10.0.7 Export position in Jellyfish .pos

writeme

10.0.8 Export position in Snowie .txt position format

writeme

10.0.9 Export positions, games, matches, and sessions in plain text

11 Settings

11.1 Player

11.2 Appearance

11.3 Options

11.4 Advanced options

11.5 Evaluation

11.5.1 Movefilters

gnubg uses a technique called "move filters" in order to prune the complete list of legal moves when analysing chequer play decisions. Move filters can be considering a generalising of the search space used in earlier versions of gnubg.

A move filter for a given ply, say, 2-ply, consists of four parameters for each subply:

1. whether to analyse at all at this subply,
2. the number of moves always accepted at the given level,
3. the number of extra moves to add,
4. the threshold for adding extra moves.

For example, for 2-ply chequer play decisions there are two movefilters: one for pruning at 0-ply, and another for pruning at 1-ply. The predefined setting "Normal" has: accept 0 moves and add up to 8 moves within 0.16 at 0-ply, and no pruning at 1-ply.

Consider the opening position where 6-2 has been rolled:

```
  GNU Backgammon  Position ID: 4HPwATDgc/ABMA
                  Match ID   : cAkZAAAAAAAA
  +13-14-15-16-17-18------19-20-21-22-23-24-+     O: gnubg
  | X           O    |   | O             X |     0 points
  | X           O    |   | O             X |
  | X           O    |   | O               |
  | X                |   | O               |
  | X                |   | O               |
 v|                  |BAR|                 |     (Cube: 1)
  | O                |   | X               |
  | O                |   | X               |
  | O           X    |   | X               |
  | O           X    |   | X             O |     Rolled 24
  | O           X    |   | X             O |     0 points
  +12-11-10--9--8--7-------6--5--4--3--2--1-+     X: jth
```

```
Pip counts: O 167, X 167
```

gnubg starts by finding all possible moves and evaluate those at 0-ply:

```
1. Cubeful 0-ply    8/4 6/4                       Eq.:  +0.207
   0.548 0.172 0.009 - 0.452 0.121 0.005
    0-ply cubeful [expert]
2. Cubeful 0-ply    13/11 13/9                    Eq.:  +0.050 ( -0.156)
   0.509 0.155 0.009 - 0.491 0.137 0.007
    0-ply cubeful [expert]
3. Cubeful 0-ply    24/20 13/11                   Eq.:  +0.049 ( -0.158)
   0.513 0.137 0.007 - 0.487 0.132 0.004
    0-ply cubeful [expert]
4. Cubeful 0-ply    24/22 13/9                    Eq.:  +0.037 ( -0.170)
   0.508 0.142 0.007 - 0.492 0.134 0.004
    0-ply cubeful [expert]
5. Cubeful 0-ply    24/22 24/20                   Eq.:  -0.008 ( -0.215)
   0.501 0.121 0.006 - 0.499 0.133 0.003
    0-ply cubeful [expert]
6. Cubeful 0-ply    24/18                         Eq.:  -0.015 ( -0.222)
   0.502 0.121 0.006 - 0.498 0.140 0.004
    0-ply cubeful [expert]
7. Cubeful 0-ply    24/20 6/4                     Eq.:  -0.023 ( -0.229)
   0.497 0.132 0.007 - 0.503 0.144 0.005
    0-ply cubeful [expert]
8. Cubeful 0-ply    13/9 6/4                      Eq.:  -0.026 ( -0.233)
   0.494 0.146 0.008 - 0.506 0.151 0.009
    0-ply cubeful [expert]
[10 moves deleted]
```

Accoring to the move filter the first 0 moves are accepted. The equity of the best move is +0.207, and according to the movefilter we add up to 8 extra moves if they're within 0.160, that is, if they have equity higher than +0.047. Move 4-18 all have equity lower that +0.047, so the move list after pruning at 0-ply consists of move 1-3. According to the move filter we do not perform any pruning at 1-ply, so move 1-3 are submitted for evaluation at 2-ply;

```
1. Cubeful 2-ply    8/4 6/4                       Eq.:  +0.197
   0.546 0.172 0.008 - 0.454 0.123 0.005
    2-ply cubeful 100% speed [world class]
2. Cubeful 2-ply    24/20 13/11                   Eq.:  +0.058 ( -0.138)
   0.515 0.141 0.007 - 0.485 0.130 0.005
    2-ply cubeful 100% speed [world class]
3. Cubeful 2-ply    13/11 13/9                    Eq.:  +0.050 ( -0.147)
   0.508 0.156 0.007 - 0.492 0.136 0.006
    2-ply cubeful 100% speed [world class]
4. Cubeful 0-ply    24/22 13/9                    Eq.:  +0.037 ( -0.159)
   0.508 0.142 0.007 - 0.492 0.134 0.004
    0-ply cubeful [expert]
5. Cubeful 0-ply    24/22 24/20                   Eq.:  -0.008 ( -0.205)
   0.501 0.121 0.006 - 0.499 0.133 0.003
    0-ply cubeful [expert]
6. Cubeful 0-ply    24/18                         Eq.:  -0.015 ( -0.212)
```

```
        0.502 0.121 0.006 - 0.498 0.140 0.004
         0-ply cubeful [expert]
     7. Cubeful 0-ply    24/20 6/4                    Eq.:  -0.023 ( -0.219)
        0.497 0.132 0.007 - 0.503 0.144 0.005
         0-ply cubeful [expert]
     8. Cubeful 0-ply    13/9 6/4                     Eq.:  -0.026 ( -0.222)
        0.494 0.146 0.008 - 0.506 0.151 0.009
         0-ply cubeful [expert]
```

If we instead request a 4-ply chequer play decision, gnubg will use the move filters defined for 4-ply:

Ply	Accept moves	Extra moves	Threshold for extra moves
0	0	8	0.160
1	no pruning		
2	0	2	0.040
3	no pruning		

The 4-ply move filter is identical to the 2-ply for pruning at 0-ply, so after 0-ply we have the same three moves as above. Since there is no pruning at 1-ply these three moves are evaluated at 2-ply as above. There is no pruning at 3-ply.

At 4-ply we do not accept any moves, but add up to two moves if there within 0.040 from the best move. Since the second best move is -0.138 worse than the best move, we do not accept any moves to be evaluated at 4-ply. Hence gnubg will actually not evaluate any moves on 4-ply.

The predefined movefilters all have “accept 0 moves”, in order to facilitate fast decisions and analysis, i.e., no need to waste much time over obvious moves.

For post-mortem analysis it may be worthwhile to ensure that gnubg analyses at least two moves at the specified ply. To do this, specify “accept 2 moves” in the move filters you use for analysis. However, do note that gnubg will force evaluation at the specified ply if the actual move made is doubtful. This ensures that all errors and blunders are evaluted at the same level.

11.6 Analysis

11.7 Rollouts

11.8 Export

12 Analysing and rollouts

12.1 Evaluating a position

12.2 Evaluating a cube decision

12.3 Rollout of moves

12.4 Rollout of cube decisions

12.4.1 Quasi-Random Dice

Quasi-Random Dice are used to reduce the element of luck in rollouts. Instead of selecting purely random dice, `gnubg` will ensure a uniform distribution of the first roll of the rollout. If 36 trials is requested, one game will start with 11, two games with 21, two games with 31, etc. In general, if n * 36 games is requested, n games will start with 11, 2*n games with 21 etc. This is called *rotation* of the first roll. Similarly, if n*1296 trials is requested, the second roll will be rotated, such that n games will start with 11-11, 2*n games with 11-21, 4*n games with 21-21, etc. The third roll be also be rotated if the number of trials is proportional to 46656.

Suppose a user stops a 1296 trial rollout after 36 games. The 36 games would have had the following rolls for the first two rolls of each game: 11-11, 21-11, 12-11, 31-11, 13-11, ..., 66-11 Obviously such a rollout will give skewed results since the second roll was 11 for all games! To avoid this problem `gnubg` will randomise the sequence of rolls such that it is guaranteed that for any sample of 36 games you have exactly one game with first roll 11, exactly one game with second roll 11, etc. This is called *stratification*.

`gnubg` will actually also rotate and stratify rollouts where the number of trials are not multiples of 36, 1296, etc. The distribution of rolls is obviously not uniform any longer in this case, but it will still provide some reduction of the luck, i.e., no 37 trial rollout will have 3 games with a initial 66.

Before the first game of a rollout, `gnubg` creates a psuedo random array which it will use for all the games in the rollout. In effect it has already decided the roll sequence it will use for up to 128 rolls in every game of the rollout. In other words, for a normal rollout where games don't go over 64 moves, every single game of every possible rollout length has already had its dice sequence determined. During the rollout of game n, sequence n will be used, for game n+1 sequence n+1, etc. If it's a *rollout as initial position*, then whenever the current sequence starts with a double, the sequence is skipped and the dice routine moves on to the next sequence. Say an *rollout as initial position* is about to start using sequence 275, but that sequence begins with a double. The dice routine moves to sequence 276. On the following game, it will use sequence 277 (it remembers how many it has already skipped).

So, if you select *rollout as initial position* and 36 games, then you will get a prefect set of rolls for games 1..30 and the first 6 rolls of the next perfect set (the same rolls you would have gotton for games 31..36 if you'd asked for 1080 games or 10800 games or 92 games or whatever.

The dice sequence doesn't know how man trials it will be asked for, it simply generates sequences such that for a normal rollout (*rollout as initial position*) every 36 (30) games you get all possible 1st rolls, every 1296 (1080) games get every possible first 2 rolls, every 46656 (38880) games you get full sets of 3 rolls, etc.

12.5 Analysing a saved game or match or session

`gnubg` allows

12.6 Output of an analysis

12.6.1 Chequer play analysis

12.6.2 Cube decision analysis

12.6.3 Game, match, or session statistics

You can get a summary of the analysis from the game, match, or session analysis. The game analysis is a summary for the current game whereas the match or session statistics is a summary of all the games in the match or session. The match analysis is available in the GUI from `Analysis->Match Statistics` or at the botton of exported files.

12.6.3.1 Chequer play statistics

This section provides a summary of the chequer play statistics. The following information is available

- **Total moves** : The total number of moves in the match.
- **Unforced moves** : The number of unforced moves, i.e., all chequer play decisions which had more than one legal move.
- **Moves marked xxx**: The number of moves marked *very good*, *very bad* etc. The analysis will mark moves *very bad*, *bad* etc. based on the threshold you've defined in the analysis settings. `gnubg` can currently not automatically mark moves, e.g., *good*, but you can mark moves *good* yourself.
- **Error rate (total)**: The first number is the total amount of normalised equity that the player gave up during this game or match. The number in parenthesis is the un-normalised counterpart; for money play the all errors are multiplied with the value of the cube, and for match play the total match winning chance given up by the player is reported.
- **Error rate (per move)**: The error rate per move is the total error rate divided by the number of *unforced moves*. Note that is different from Snowie 4 that defines the error rate per move as the total error rate divided by the total number of moves for both players, i.e., the sum of *total moves* for both players. In general, your error rate per move will be lower in Snowie than in `gnubg`.

- **Chequer play rating**: gnubg will assign a rating for your chequer play ranging from *Awful!* to *Supernatural*. See the description for the *overall rating* below.

12.6.3.2 Luck analysis

This section provides information about how Ms. Fortuna distributed her luck. The following information is available:

- **Rolls marked xxx**: The number of moves marked *very lucky*, *lucky* etc. Moves marked *very lucky* are huge jokers that improve your equity with more the +0.6 relative to the average equity. gnubg normally uses cubeful 0-ply evaluations to calculate the luck, but you can change that under the analysis settings. See below for a complete overview of what is considered *very lucky*, *lucky*,etc.
- **Luck rate (total)**: The total luck for this game or match reported both normalised and unnormalised.
- **Luck rate (per move)**: The luck rate per move reported both normalised and unnormalised.
- **Luck rating**: Based on the luck rate per move gnubg will assign you a humerous luck rating. See below for the possible ratings.

Thresholds for marking of rolls:

Deviation of equity from average	**Roll is marked**
>0.6	*very lucky*
0.3 - 0.6	*lucky*
-0.3 - 0.3	unmarked
-0.6 - -0.3	*unlucky*
< -0.6	*very unlucky*

Luck ratings:

Normalised luck rate per move	**Luck rating**
> 0.10	Cheater :-)
0.06 - 0.10	Go to Las Vegas immediately
0.02 - 0.06	Good dice, man!
-0.02 - 0.02	none
-0.06 - -0.02	Better luck next time
-0.06 - -0.10	Go to bed
< -0.10	Haaa-haaa

12.6.3.3 Cube statistics

This section provides a summary of the cube decision statistics: the number of cube decisions, missed doubles, etc.

- **Total cube decisions**: The total number of cube decisions, i.e., the sum of no-doubles, doubles, takes, and passes.
- **Close or actual cube decisions**: Similar to "Total cube decisions", except that no-doubles are only included if they're considered "close". gnubg considers a cube decision close if the relavant equities are within 0.25 from eachother or if the position is too good.

- **Doubles, Takes, Passes**: The total number of doubles, takes, and passes in the game or match.
- **Missed doubles around DP, Missed doubles around TG**: The number of missed doubles around the double point and too good point respectively. If the equity is below 0.95 the position is considered to be around the double point else it considered to be around the too good point. If you miss doubles around the double point, it usually means that you double to late, whereas missed doubles around the too good point means that you're too greedy and play on for gammon when the position is *not* too good.
- **Wrong doubles around DP, Wrong doubles around TG**: The number of wrong doubles around the double point and too good point, respectively. If you have missed doubles around the double point that generally means that you double too early where you really should hold the cube, whereas missed doubles around the too good point means that you double positions where you really should play on for a gammon or backgammon.
- **Wrong takes, Wrong passes**: The total number of wrong takes and passes, respectively.
- **Error rate (total)**: The accumulated cube errors for this game or match reported both normalised and unnormalised.
- **Error rate (per cube decision)**: The error rate per cube decision is the total error rate divided by the number of *close or actual cube decisions*. Note that is different from Snowie 4 that defines the error rate per cube decision as the total error rate divided by the total number of moves for both players. In general, your error rate per cube decision will be lower in Snowie than in `gnubg`.
- **Cube decision rating**: `gnubg` will assign a rating for your cube decisions ranging from *Awful!* to *Supernatural*. See the description for the *overall rating* below.

12.6.3.4 Overall rating

The last section is the overall summary.

- **Error rate (total)**: The accumulated chequer play and cube decision errors reported by normalised and unnormalised.
- **Error rate (per decision)**: The error rate per decision is the total error rate divided by the number of non-trivial decisions (i.e., the sum of *unforced moves* and *close or actual cube decisions*). Again, please note that is different from Snowie 4 that defines the error rate per decision as the total error rate divided by the total number of moves for both players. In general, your error rate per move will be lower in Snowie than in `gnubg`. An investigation of approximately 300 matches showed the on average the `gnubg` error rate with be 1.4 times higher than your Snowie 4 error rate.
- **Equivalent Snowie error rate**: For easy comparison `gnubg` will also print the total error rate calculated by the same formulae as Snowie 4.
- **Overall rating**: Based on your normalised error rate per move `gnubg` will assign you a rating ranging from *Awful!* to *Supernatural*. See the table below for the thresholds.
- **Actual result**: The actual result of the game or match. For money game this is simply the number of points won or lost during the game or match. For match play the number is calculated as 50% added to the result of the game or match in MWC. For example, losing a match or winning a match corresponds to an actual result of 0% and 100%,

respectively. Winning 1 point in the first game of 7 point match is worth 6% using Kit Woolsey's match equity table, hence the actual result is 56%.

- **Luck adjusted result**: The luck adjusted result is calculated as the *actual result* plus the total unnormalised luck rate. This is also called "variance reduction of skill" as described in Douglas Zare's excellent article Hedging Toward Skill (`http://math.columbia.edu/~zare/vrskill.html`). This should give an unbiased measure of the strengths of the players.
- **MWC against current opponent**: For match play `gnubg` will calculate your MWC against the current opponent. The number is calculated as 50% - your total unnormaluised error rate + your opponent's total unnormalised error rate. If your opponent is really lucky but plays rather bad, this number can become larger than 100%, since he due to the extra luck has the possibily to give up even more MWC. However, `gnubg` will report this number as 100%, and 0% for the opposite situation. Note that this number is biased towards the analysing bot, e.g., a 0-ply analysis a game between `gnubg` 2-ply and 0-ply will suggest that 0-ply is a favourite, which it is clearly not. For an unbiased measure use the *luck adjusted result* above.
- **Guestimated abs. rating**: writeme

Threshold for ratings:

Normalised total error rate per move	**Rating**
0.000 - 0.002	Supernatural
0.002 - 0.005	World Class
0.005 - 0.008	Expert
0.008 - 0.012	Advanced
0.012 - 0.018	Intermediate
0.018 - 0.026	Casual Player
0.026 - 0.035	Beginner
> 0.035	Awful!

12.7 Equities

12.7.1 Introduction

`gnubg` works with many different kinds of equities. The equity is defined as the expected value of the position. However, this value can be expressed in several different metrics and may be calculated with or without taking the effect of the cube into consideration. In the following section we will describe the equities used and calculated by `gnubg`.

12.7.2 Money equity

This is the value of the position in money game, e.g., if you equity is +0.4 an you are playing money game with a $1 stake, you will win $0.40 on average. The money equity can be calculated with or without taking the effect of the doubling cube into consideration, or *cubeful* or *cubeless*. The cubeless equity can be calculated from the basic formulae: 2*p(w)-1+2(p(wg)-p(lg))+3(p(wbg)-p(lbg)). Evaluating the cubeful equity is much more difficult; it can either be estimated from the cubeless equitity by using transformations as outlined by Rick Janowski or by constructing a neural net that directly outputs cubeful equities. `gnubg` uses the former approach (see chapter ⟨undefined⟩ [Cubeful equities], page ⟨undefined⟩).

12.7.3 Match Winning Chance

In match play we're generally not particular interested in the outcome of the individual games as much as the outcome of the entire match, so the interesting quantity for match play is *match winning chance* (MWC). As for the money equity the MWC can be calculated with and without the effect of the doubling cube. The MWCs are generally calculated with the use of a match equity table, which contains the chance of winning the match *before* a game starts, e.g., if the score is 0-0 in a 1pt match each player has 50% chance of winning the match before the game starts assuming they're of equal skill.

The cubeless MWC is calculated as: MWC(cubeless) = p(w) * MWC(w) + p(l) * MWC(l) + p(wg) * MWC(wg) + p(lg) * MWC(lg) + p(wbg) * MWC(wbg) * p(lbg) * MWC(lbg).

For example, if the w/g/bg distribution is 0 30 60 - 40 10 0 and the match score is 1-3 to 5 with the cube on 2 the cubeless MWC is:

MWC(cubeless)= 30% * 50% + 30% * 0% + 30% * 100% + 10% * 0% + 0% * 100% + 0% * 0% = 45%,

so the cubeless MWC is 45%.

Evaluating the cubeful MWC is more difficult, and as for the cubeful money equity it's possible to estimate cubeful MWCs from transformation on the w/g/bg distribution or directly calculate it from neural nets. `gnubg` uses the former approach, but the formulae are currently not published.

12.7.4 Normalised equity

It's generally very difficult to compare MWCs. For example, it's hardly worth mentioning a 0.5% MWC error at DMP where as it's a huge error at 0-0 to 7. It is therefore of interesting to normalise the MWCs to some common scale. The most often used normalisation is Normalised Money Game Equity (NEMG) where the MWC for any game is transformed into the same interval as money game, i.e., -3 to +3 (due to anomalies at certain match scores the NEMG can go below -3 and above +3). The transformation is linear:

NEMG = 2 * (MWC-MWC(l))/(MWC(w)-MWC(l)) - 1

In other words, extrapolation with the following two extrapolation points: (MWC(w),+1) and (MWC(l),-1).

For example, suppose the score is 3-1 to 5 with the cube on 2: MWC(l)=0% and MWC(w)=50%:

MWC	**NEMG**
0%	-1
25%	0
50%	+1
75%	+2
100%	+3

Note that a w/g/bg distribution of 0 100 100 - 0 0 0 gives a NEMG of +3 whereas the corresponding money equity is only +2. This is because the gammon price is high for that particular score. When both players are far from winning the match, e.g., 0-0 to 17 or 1-0 to 17, NEMG is very close to the ususal money equity.

NEMG can be calculated from both cubeless and cubeful MWCs.

A word of caution: A cubeless NEMG calculated from a cubeless MWC could be named "cubeless equity", but in most backgammon litterature this term seems to be reserved for the cubeless money equity.

13 Position and match IDs

13.1 A technical description of the position ID

13.1.1 Introduction

This section describes a method for compactly recording a backgammon position. It demonstrates how to encode a position into 10 binary bytes, which is useful for minimising the space used when recording large numbers of positions in memory or on disk. There is also an ASCII representation in 14 characters, which is convenient for output to the screen, for copying and pasting to transfer positions between programs which support the format, and for communicating positions via Usenet news or e-mail. The 10 byte binary format is called the *key*, and the 14 character ASCII format is the *ID*.

13.1.2 Key format

The key is essentially a bit string (imagine you start with an empty sequence of bits, and continue adding either "0" or "1" to the end). The way to build up a sequence that corresponds to a given position is:

1. For every point around the board (starting at the ace point of the player on roll, continuing around to the 24 point and ending at the bar):
   1. append as many 1s as the player on roll has on that point (if any).
   2. append a 0.
2. For every point around the board (starting at the ace point of the opponent, continuing around to the opponent's 24 point and ending at the bar):
   1. append as many 1s as the opponent has on that point (if any).
   2. append a 0.
   3. Pad out the string to 80 bits with 0s.

The worst-case representation will require 80 bits: you can see that there are always 50 0 bits even if there are no chequers at all. Each player has a maximum of 15 chequers in play (not yet borne off) which require a 1 bit wherever they are positioned. That's 30 bits to take of all chequers, plus the 50 bits of overhead for a total of 80 bits (the last bit is always 0 and isn't strictly necessary, but it makes the code slightly easier). This bit string should be stored in little-endian order when packed into bytes (ie. the first bits in the string are stored in the least significant bits of the first byte).

As an example, here's what the starting position looks like in the key format:

0 0 0 0 0	player on roll has no chequers on ace to 5 points
11111 0	5 chequers on the 6 point
0	empty bar
111 0	3 on the 8
0 0 0 0	no others in our outfield
11111 0	5 on the midpoint

0 0 0 0 0	none in the opponent's outfield
0 0 0 0 0	or in opponent's board, until...
11 0	two on the 24 point
0	none on the bar
0 0 0 0 0	opponent has no chequers on ace to 5 points
11111 0	5 chequers on the 6 point
0	empty bar
111 0	3 on the 8
0 0 0 0	no others in opponent's outfield
11111 0	5 on the midpoint
0 0 0 0 0	none in our outfield
0 0 0 0 0	or in our board, until...
11 0	two on the 24 point
0	none on the bar

so altogether it's:

00000111110011100000111110000000000011000000011111001110000011111000000000001100

In little endian bytes it looks like:

11100000 01110011 11110000 00000001 00110000 11100000 01110011 11110000 00000001 00110000 0xE0 0x73 0xF0 0x01 0x30 0xE0 0x73 0xF0 0x01 0x30

so the 10 byte key (in hex) is E0 73 F0 01 30 E0 73 F0 01 30. ID format

The ID format is simply the Base64 (`http://www.ietf.org/rfc/rfc2045.txt`) encoding of the key. (Technically, a Base64 encoding of 80 binary bits should consist of 14 characters followed by two = padding characters, but this padding is omitted in the ID format.)

To continue the above example, splitting the 10 8-bit bytes into 14 6-bit groups gives:

111000 000111 001111 110000 000000 010011 000011 100000 011100 111111 000000 000001 001100 000000

In Base64 encoding, these groups are respectively represented as:

4 H P w A T D g c / A B M A

So, the position ID of the chequers at the start of the game is simply:

4HPwATDgc/ABMA

You can set the board in `gnubg` either by writing the position ID into the position text input field in the GUI or by executing the command `set board 4HPwATDgc/ABMA`.

13.1.3 Notes

This encoding is obviously not as compact as it could be: in particular, there are lots of redundant representations of illegal positions where both players have chequers on the same point. Theoretically, it would be possible to get it down to 64 bits by using Walter Trice's "D() expressions" (`http://www.bkgm.com/rgb/rgb.cgi?view+371`), but I think you'd have to be a mathematical masochist to try it!

Example code (`http://savannah.gnu.org/cgi-bin/viewcvs/gnubg/gnubg/positionid.c?rev=HEAD&s`
to convert between a raw board encoding (the number of chequers on each point) and these keys/IDs is available licensed under GPL.

Thanks to Tom Keith and David desJardins for their suggestions on simplifying the encoding without increasing the worst case length.

13.2 A technical description of the match ID

13.2.1 Introduction

This section describes how the match ID is calculated. The match ID can be used for easy exchange of positions for `gnubg` users in conjuction with the position ID. The match key is a 9 byte representation of the match score, match length, value of cube, owner of cube, Crawford game flag, player on roll, player to make a decision, doubled flag, resigned flag, and the dice rolled. The match ID is the 12 character Base64 encoding of the match key.

13.2.2 Match key

The match key is a bit string of length 66:

1. Bit 1-4 contains the 2-logarithm of the cube value. For example, a 8-cube is encoded as 0011 binary (or 3), since 2 to the power of 3 is 8. The maximum value of the cube in with this encoding is 2 to the power of 15, i.e., a 32768-cube.
2. Bit 5-6 contains the cube owner. 00 if player 0 owns the cube, 01 if player 1 owns the cube, or 11 for a centered cube.
3. Bit 7 is the player on roll or the player who did roll (0 and 1 for player 0 and 1, respectively).
4. Bit 8 is the Crawford flag: 1 if this game is the Crawford game, 0 otherwise.
5. Bit 9-11 is the game state: 000 for no game started, 001 for playing a game, 010 if the game is over, 011 if the game was resigned, or 100 if the game was ended by dropping a cube.
6. Bit 12 indicates whose turn it is. For example, suppose player 0 is on roll then bit 7 above will be 0. Player 0 now decides to double, this will make bit 12 equal to 1, since it is now player 1's turn to decide whether she takes or passes the cube.
7. Bit 13 indicates whether an doubled is being offered. 0 if no double is being offered and 1 if a double is being offered.
8. Bit 14-15 indicates whether an resignation was offered. 00 for no resignation, 01 for resign of a sigle game, 10 for resign of a gammon, or 11 for resign of a backgammon. The player offering the resignation is the inverse of bit 12, e.g., if player 0 resigns a gammon then bit 12 will be 1 (as it is now player 1 now has to decide whether to accept or reject the resignation) and bit 13-14 will be 10 for resign of a gammon.
9. Bit 16-18 and bit 19-21 is the first and second die, respectively. 0 if the dice has not yet be rolled, otherwise the binary encoding of the dice, e.g., if 5-2 was rolled bit 16-21 will be 101-010.

10. Bit 22 to 36 is the match length. The maximum value for the match length is 32767. A matchscore of zero indicates that the game is a money game.
11. Bit 37-51 and bit 52-66 is the score for player 0 and player 1 respectively. The maximum value of the match score is 32767.

For example, assume the score is 2-4 in a 9 point match with player 0 holding a 2-cube, and player 1 has just rolled 52. The match key for this will be (note that the bit order is reversed below for readability)

1000 00 1 0 100 1 0 00 101 010 100100000000000 010000000000000 001000000000000

In little endian the bytes looks like:

01000001 10001001 00101010 00000001 00100000 00000000 00100000 00000000 00

0x41 0x89 0x2A 0x01 0x20 0x00 0x20 0x00 0x00

13.2.3 Match ID

Analogous to the position ID from the previous section the match ID format is simply the Base64 encoding of the key.

To continue the example above, the 9 8-bit bytes are grouped into 12 6-bits groups:

010000 011000 100100 101010 000000 010010 000000 000000 001000 000000 000000 000000

In Base64 encoding, the groups are represented as:

Q Y k q A S A A I A A A

So, the match id is simply:

QYkqASAAIAAA

If someone post a match ID you can set up the position in `gnubg` by writing or pasting it into the Match ID text input field on the main window, or by executing the command `set matchid QYkqASAAIAAA`.

14 Cubeful equities

This chapter is a brief description of how gnubg calculates cubeful equities. The formulae build directly on the work by Rick Janowski Take-Points in Money Games (http://www.msoworld.com/mindzine/news/classic/bg/cubeformulae.html) from 1993.

14.1 Basic formulae for cubeful equities

The basic formulae for cubeful equities as derived by Janowski is

$$E(cubeful) = E(dead) \cdot (1 - x) + E(live) \cdot x.$$

where E(dead) is the dead cube equity (cubeless equity) calculated from the standard formulae. E(live) is the cubeful equity assuming a fully live cube. We'll return to that in the next section. x is the cube efficiency. x=0 gives E(cubeful)=E(dead) as one extreme and x=1 gives E(cubeful)=E(live) as the other extreme. In reality x is somewhere in between, which typical values around 0.6 - 0.8.

Janowski's article doesn't mention cubeful equities, so we use the straightforward generalisation

$$MWC(cubeful) = MWC(dead) \cdot (1 - x) + MWC(live) \cdot x.$$

as MWC is the entity that is used for match play evaluations.

14.2 Live cube equities

The live cube equity is the equity assuming that the equity changes continuously, so that doubles and takes occurs exactly at the double point and take point. For gammonless play this is the well-known take point of 20%. Janowski derives the more general formula

$$TP = L - 0.5W + L + 0.5$$

where W is the average cubeless value of games ultimately won, and L is the average cubeless value of games ultimately lost. For example, for the following position

```
 GNU Backgammon  Position ID: 4HPMwQCMz+AIIQ
                 Match ID   : cAkAAAAAAAAA
 +13-14-15-16-17-18------19-20-21-22-23-24-+     O: gnubg
 | X  O     X  O    |   | O  X           X |     0 points
 | X  O        O    |   | O              |
 | X           O    |   | O              |
 |                  |   | O              |
 |                  |   | O              |
v|                  |BAR|                |     (Cube: 1)
 |                  |   | X              |
 |                  |   | X              |
 | O                |   | X              |
```

```
| O            X  O |   | X        X        |     On roll
| O            X  O |   | X        X        |     O points
+12-11-10--9--8--7-------6--5--4--3--2--1-+     X: jth
```

gnubg evaluates

```
          Win    W(g)   W(bg)  L(g)   L(bg)
static: 0.454   0.103  0.001  0.106  0.003
```

and hence W=(0.454 + 0.103 + 0.001)/0.454=1.229 and L=(0.556+0.106+0.003)/0.556) = 1.196. For gammonless positions, e.g., a race, W=1 and L=1.

The live cube equity is now based on piecewise linear interpolation between the points (0%,-L%), (TP,-1), (CP,+1), and (100%,+W): if my winning chance is 0 I lose L points, at my take point I lose 1 point, at my cash point I cash 1 point, and when I have a certain win I win W points:

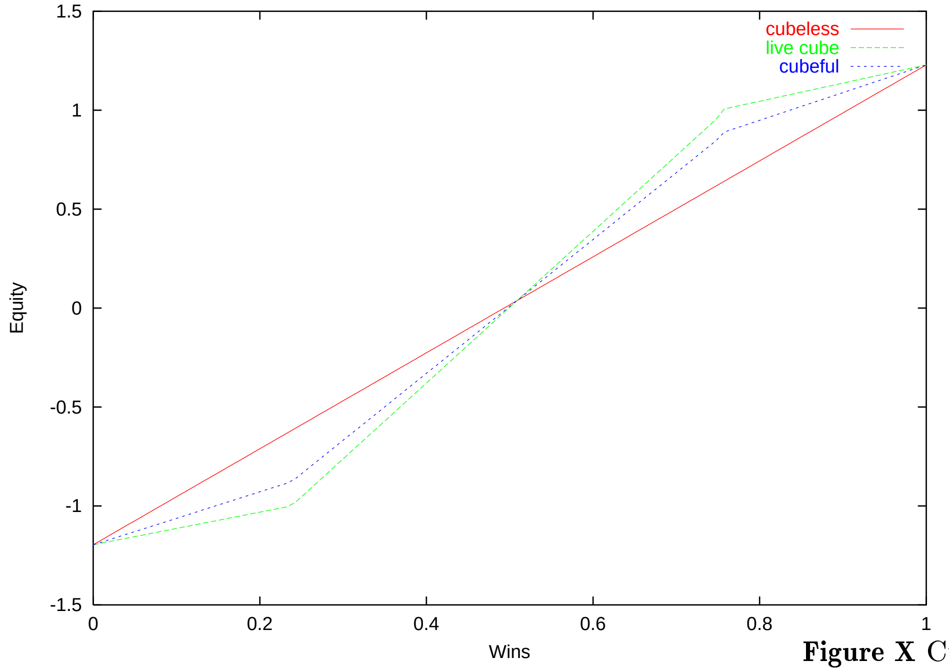

Figure X Cubeful equities

For match play there is no simple formula, since redoubles can only occur a limited number of times.

The live cube take point is generally calculated as

TP(live, n Cube)=TP(dead, n cube) * (1 - TP(live, 2n cube)

So to calculate the live cube take points for a 1-cube at 3-0 to 7 we need the live cube take points for the 4-cube and the 2-cube. For the position above and using Woolsey's match equity table the live cube take point are:

Cube value	**TP for X**	**TP for O**
4	0%	41%
2	15%	38.5%
1	24.5%	27.3%

The calculation of these are left as an exercise to the reader.

Ignoring backgammons, the gammon rates for O and X are 0.106/54.6=19% and 0.103/0.454=22%, respectively. If O wins the game his MWC will be

81% * MWC(-3,-7) + 19% * MWC(-2,-7) = 78%

and if X wins his MWC will be

78% * MWC(-4,-6) + 22% * MWC(-4,-5) = 41%.

If O cashes 1 point he has MWC(-3,-7)=76% and if X cashes he has MWC(-4,-6)=36%. Analogous to money game the live cube MWC is calculated as piecewise linear interpolation between (0%,22%), (24.5%,24%), (72.7%,36%), and (100%,41%) (from X's point of view):

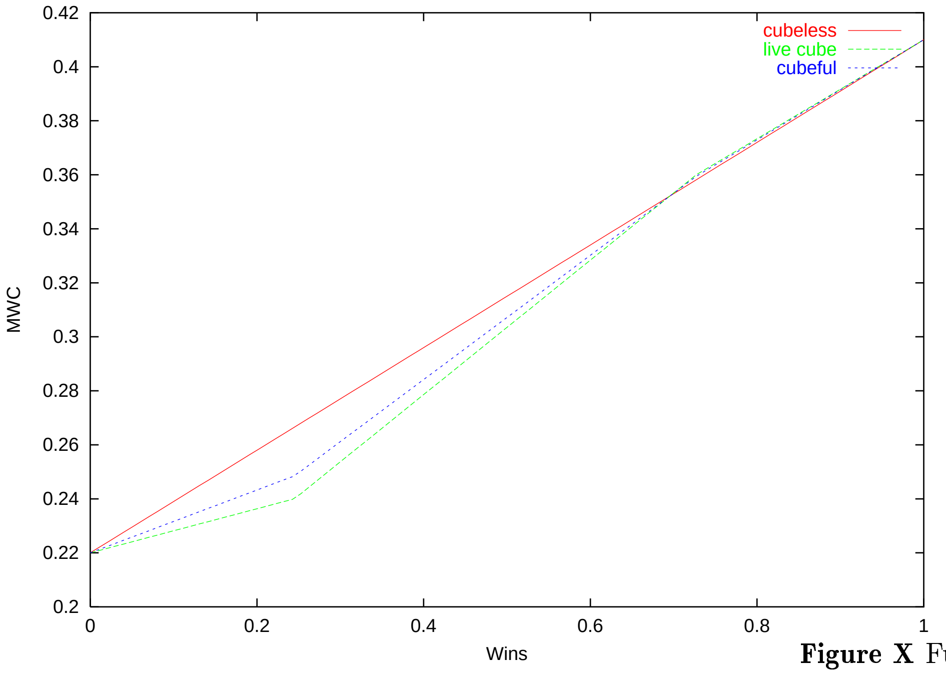

Figure X Fully live cubeful MWC

14.3 0-ply Cubeful equities

Having established the live cube equities and MWCs we're now in position to calculate the 0-ply cubeful equities.

Let's start with money game: the cubeless equity is -0.097 and the live cube equity can be determined from Figure X as -0.157. Thus, the cubeful equity is -0.138.

For the match play example at the score 3-0 the cubeless MWC is 29.1% and from Figure X using wins=45.4% we can determine the live cube MWC to be 29.2%. Using a value of x=0.68 we arrive at a cubeful MWC of 29.17%.

14.4 n-ply Cubeful equities

The previous section concerned the calculation of 0-ply cubeful equities, so how so `gnubg` calculate cubeful 2-ply equities? The answer is: by simple recursion:

```
Equity=0
Loop over 21 dice rolls
```

```
    Find best move for given roll
    Equity = Equity + Evaluate n-1 ply equity for resulting position
  End Loop
  Equity = Equity/36
```

Note that evaluating the n-1 ply equity involves a cube decision, since the opponent may double, so gnubg will actually calculate the two n-1 ply equities: (a) assuming no double, and (b) assuming double, take. These two equities are combined with the equity for a pass, and the optimum of these three is added to the resulting equity. For a cubeful 2-ply evaluation gnubg will end up calculating the following cubeful 0-ply equities: centred 1-cube, opponent owns 2-cube, owned 4-cube, and opponent owns 8-cube.

Note that the 2-ply level does not use the cube efficiency, it's not used until at the 0-ply level, but it's possible to calculate an effective one by isolating x in the basic cube formulae:

x(eff) = (E(2-ply cubeful) - E(2-ply dead))/(E(2-ply live)-E(2-ply dead)).

14.5 The cube efficiency

The cube efficiency is obviously an important parameter, unfortunately there haven't been much investigation carried out, so gnubg basically uses the values 0.6-0.7 originally suggested by Rick Janowski:

Position Class	**x (Cube efficiency)**
Two-sided (exact) bearoff	n/a
One-sided bearoff	0.6
Crashed	0.68
Contact	0.68
Race	linear interpolation between 0.6 and 0.7

For race gnubg uses linear interpolation based on pip count for the player on roll. A pip count of 40 gives x=0.6 and 120 gives x=0.7. If the pip count is below 40 or above 120 values of x=0.6 and x=0.7 are used, respectively.

For the two sided bearoff positions the cubeful money equity is already available from the database, so for money game there is no need to calculate cubeful equities via Janowski's formulae. However, the cubeful equities for money game cannot be used for match play. Instead of using a fixed value of x, say, 0.6, gnubg will calculate an effective value based on the cubeful money equity. The cubeful MWC is calculated as usual, but with the calculated x.

There is obviously room for improvements. For example, holding games should intuitively have a lower cube efficiency, since it's very difficult to double effectively: either it's not good enough or you've lost the market by a mile after rolling a high double or hitting a single shot. Similarly, backgames will often have a low cube efficiency, whereas blitzes have may have a higher cube efficiency.

14.6 Cube decisions

gnubg's cube decisions are simple based on calculations of cubeful equities. For a double decision gnubg calculates the cubeful equity for "no double" and the cubeful equity for “double, take”. Combined with the equity for “double, pass”, it’s possible to determine the correct cube action.

Figure X shows the relevant cubeful equities for O and X’s cube decisions in sample position from earlier.

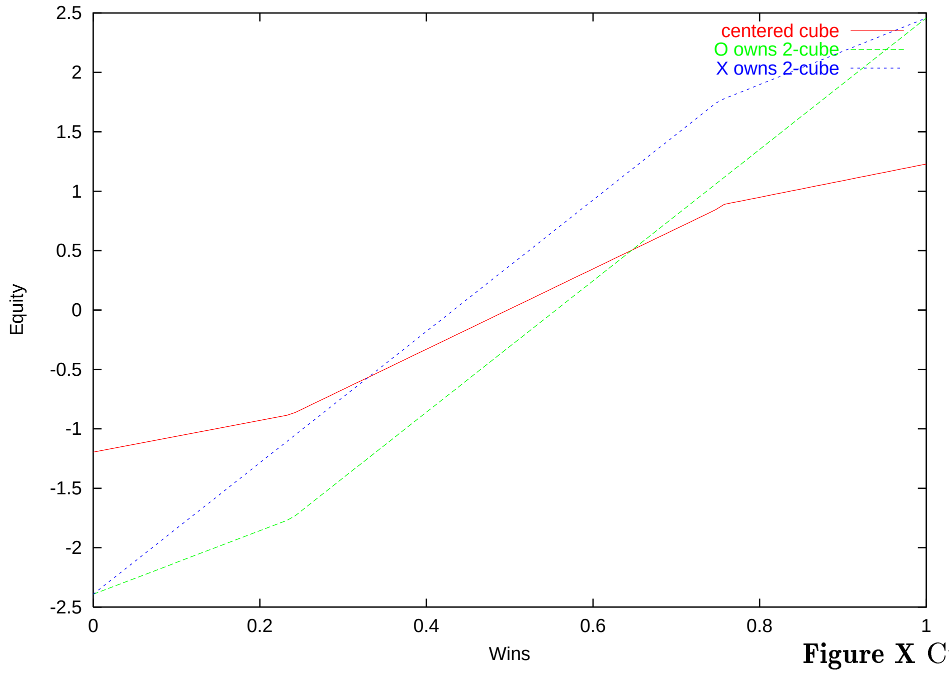

Figure X Cubeful equities

On 0-ply X will double when the green curve (O owns 2-cube) is above the red curve (centered cube), and O will take as long as the green curve is below 1. Similarly, O will double when the blue curve (X owns 2-cube) is below the red curve (centered cube), and X takes as long as the blue curve is above -1.

Note that `gnubg` doesn’t calculate the take point or double point explicitly. The cube decision is simply made by comparing equities from Figure X.

14.7 Beyond the simple model

Janowski has developed two other models for cubeful equities. The first is a generalisation of the one used by `gnubg`; it introduces two cube efficiencies instead of one. Often you may see that the cube efficiencies are different for the two players, and the “refined general model” as it is named by Janowski, tries to take this into consideration by using different cube efficiency parameters for the two players. For example, the blitzer may have another cube efficiency that the blitzee.

The second model is not published, but redefines the cube efficiency into a value that can be understood more intuitively and calculate easily from rollouts.

15 Training

16 Scripting

16.1 Accessing gnubg Python shell

To access the Python shell, either type '`>`' from the command line or select `Windows->Python Shell(IDLE...)` from the GUI.

16.2 gnubg module functions

`board()`

`command(cmd)`
`evaluate()`
`evalcontext()`
`eq2mwc()`

`mwc2eq()`

`cubeinfo()`
`met()`

`positionid()`
`positionfromid()`
`positionkey()`
`positionfromkey()`
`positionbearoff()`
`positionfrombearoff()`
`navigate([next=`*N*`,[game=`*N*`]])`

Match navigation.

Without any arguments, go to first move of first match.

With next == *N*, move forward *N* game records.

With game == *N*, move forward/backward *N* games.

Navigate never wraps around.

On success, returns `None`. If unable to complete the requsted number of moves, returns a pair of `(next-remaining,game-remaining)`.

`match([analysis=1/0, boards=1/0, statistics=0/1, verbose=0/1])`

Return the current ⟨undefined⟩ [match], page ⟨undefined⟩. For example,

```
> m = gnubg.match()
```

Takes the following optional keyword arguments:

`analysis` When 0, discard analysis data. default is 1.

`boards` When 1, add current board to move/double records. Default is 1.

`statistics`
When 1, include game and match statistics. Default is 0.

`verbose` When 1, include derived analysis values. Default is 0.

16.2.1 Match description

`gnubg.match()` returns a dictionary containing the following items:

`match-info`
: General match info. ⟨undefined⟩ [match-info], page ⟨undefined⟩

`games`
: A sequence, one elemet per game. ⟨undefined⟩ [game], page ⟨undefined⟩

`stats (optional)`
: Match statistics.

16.2.1.1 Match info

A dictionary containing the following items:

`match-length`
`variation`
: One of `Standard`,`Nackgammon`, `Hypergammon1`, `Hypergammon2` or `Hypergammon3`.

`rules (optional)`
: Additional rules used. A subset of `NoCube`, `Crawford` and `Jacoby`.

`X`
`O`
: Per player information. Each a dictionary containing `rating` and `name`.

`annotator (optional)`
`round (optional)`
`place (optional)`
`date (optional)`
: Sequence of (Day,Month,Year).

`event (optional)`
`default-eval-context`
: Default evaluation context. A dictionary in the same format as returned by `evalcontext()`.

`default-rollout-context`
: Default rollout context.

Example,

```
>>> m['match-info']
{'match-length': 25, 'rules': ('Crawford',), 'default-eval-context': {'plies': 2, 'det
```

16.2.1.2 Python game

A dictionary containing the following items:

`info`
: General game info. For example,

```
>>> m['games'][0]['info']
{'points-won': 1, 'score-X': 0, 'score-O': 0, 'winner': 'X', 'resigned': Fal
```

If no winner is specified, `winner` is `None`.

```
>>> m['games'][2]['info']
{'score-X': 2, 'winner': None, 'score-O': 0}
```

`game` A Sequence of actions. ⟨undefined⟩ [actions], page ⟨undefined⟩

`stats (optional)`
Game statistics. Similar entries to `Analyse->Game statistics` from the GUI.

16.2.1.3 Game actions

Each action is a dictionary

17 Bearoff databases

This section describes what kind of bearoff databases you can use with `gnubg` as well as how you may obtain these.

17.1 Introduction to bearoff databases

There are two kind of bearoff databases: two-sided (exact) bearoff databases or one-sided (approximative) bearoff databases.

17.1.1 Two-sided bearoff databases

Two-sided bearoff databases contain exact probabilities or equities for winning.

For example, for less than 6 chequers inside the home quadrant, each side has 924 different positions, for a total of 924 * 924 = 853,776 positions possible. The bearoff database will contain the exact winning probability for each of these 853,776 positions. Typically, the database also includes cubeful equities for money game. Cubeful equities for match play is generally not included as it is dependent on match score and cube level.

Consider the following position:

```
 GNU Backgammon  Position ID: CgAAEAEAAAAAAA
                 Match ID   : cIkMAAAAAAAA
 +13-14-15-16-17-18------19-20-21-22-23-24-+     O: gnubg
 |                  |   |          O  O    | OOO 0 points
 |                  |   |                  | OOO
 |                  |   |                  | OOO
 |                  |   |                  | OO
 |                  |   |                  | OO
v|                  |BAR|                  |     (Cube: 1)
 |                  |   |                  | XX
 |                  |   |                  | XX
 |                  |   |                  | XXX
 |                  |   |                  | XXX On Roll
 |                  |   |    X        X    | XXX 0 points
 +12-11-10--9--8--7-------6--5--4--3--2--1-+     X: jth
```

Using a two-sided bearoff database will yield that player X has 67.1% cubeless chance of winning. The database also gives cubeful money equities of +0.3441, +0.3210, and +0.1605 for X owning cube, centred cube, and O owning cube, respectively. So it's an initial double since +0.3210 >= 2 * +0.1605. However it's not a redouble since +0.3441 >= 2 * 0.1605.

The major problem with two-sided databases is that the size increases incredible fast with the number of points and chequers:

Chequers	Points	Number of positions
6	6	853,776
7	6	2,944,656
8	6	9,018,009
...		
15	6	2,944,581,696

...
15 24 18,528,584,051,601,162,496

`gnubg` stores the equity for each position with a precision of approximately 0.00003 which requires 2 bytes of storage. Hence, storing one cubeless and three cubeful equities requires a total of 8 bytes per position. This gives the following storage requirements for n chequers inside the home quadrant:

Chequers	**Points**	**Number of positions**	**Size/MB**
6	6	853,776	6
7	6	2,944,656	22
8	6	9,018,009	68
9	6	25,050,025	191
10	6	64,128,064	489
11	6	153,165,376	1,168
12	6	344,622,096	2,629
13	6	736,145,424	5,616
14	6	1,502,337,600	11,461
15	6	2,944,581,696	22,465

For the typical user the limit is probably 10 or 11 chequers unless she owns vast amounts of disk space. Also, the time to generate these databases increases proportionally with the size. The 15 chequer database takes at least 3,500 times the time it takes to generate the 6 point database. For example, on the author's 1GHz Pentium III it takes approximately 6 minutes to generate the 6 points database, hence it would take at least 21,000 minutes or approximately 15 days to generate the 15 point database on the same computer.

17.1.2 One-sided bearoff databases

Instead of looking at both player's positions simultaneously large savings can be obtained by looking at each side independently. For each position we tabulate the probability P(n) what the player will bear all chequers off in n rolls; P(n) is the one sided bearoff distribution. Assume player 0 and player 1 has one sided bearoff distributions P0(n) and P1(n), respectively. The chance of player 0 bearing all chequers off before player 1 is:

$$p = \sum_{i=0}^{\infty} P_0(i) \left(\sum_{j=i}^{\infty} P_1(j) \right)$$

For example, consider the following position:

```
 GNU Backgammon  Position ID: 2x0AAOi2AQAAAA
                 Match ID   : cAkAAAAAAAAA
 +13-14-15-16-17-18------19-20-21-22-23-24-+     O: gnubg
 |                  |   |       O  O  O  O | O   0 points
 |                  |   |       O  O  O  O | O
 |                  |   |       O  O       | O
 |                  |   |                  | O
 |                  |   |                  | O
v|                  |BAR|                  |     (Cube: 1)
```

```
   |                 |  |                 | X
   |                 |  |                 | X
   |                 |  |              X  | X
   |                 |  |  X  X  X  X     | X   On roll
   |                 |  |  X  X  X  X  X  | X   0 points
  +12-11-10--9--8--7-------6--5--4--3--2--1-+   X: jth
```

The one sided bearoff distributions are

Rolls	jth	gnubg
3	1.917%	2.811%
4	18.749%	28.403%
5	44.271%	50.307%
6	32.998%	18.114%
7	2.029%	0.363%
8	0.037%	0.002%

Applying the formula above gives:

$1.917\% \times 100\% + 18.749\% \times 97.189\% + 44.271\% \times 68.786\% \ldots + 0.037\% \times 0.02\% = 56.7\%$

The cubeless gwc's calculated from one sided bearoff distributions are usually quite good, although no thorough investigations have been performed so far. Although the databases are one sided they can also give correct moves in "desperation" scenarios.

By storing the probability to bearoff at least one chequer, it's also possible to calculate gammon probabilities, as the chance of being gammoned is the probability that my opponent bears all chequers off before I bear at least one chequer off. Analogously, it's possible to calculate the chance of gammoning the opponent.

The storage requirements are much smaller than for the equivalent two sided databases.

Chequers	Points	Number of positions
15	6	54,264
15	7	170,544
15	8	490,314
15	9	1,307,504
15	10	3,268,870
15	11	7,726,160
15	12	17,383,860
15	13	37,442,160
15	14	77,558,760
15	15	155,117,250
15	16	300,540,195
15	17	565,722,720
15	18	1,037,158,320

For example, 15 chequers on 6 points is only 54,264 positions for the one sided bearoff database compared to 294,458,696 positions for the equivalent two sided bearoff database. However, for each position we need to store an array of probabilities. For example, for 15 chequers on the 18 point we have to store 15 non-zero probabilities compared to only one in the two sided bearoff database. The table below gives approximate database sizes for `gnubg`'s one sided databases (both bearoff and gammon distributions):

Chequers	Points	Size in MB

15	6	1.4
15	7	5
15	8	15
15	9	42
15	10	112
15	11	284
15	12	682
15	13	1,543
15	14	approx 4,300
15	15	approx 10,700
15	16	approx 26,700
15	17	approx 66,700
15	18	approx 166,000

So, the practical limit is probably around the 11 to 13 point, depending on the disk space available.

`gnubg` can generate one sided bearoff database where the exact bearoff distribution is approximated by a normal distribution: instead of storing up to 15 or 20 non-zero probabilities only two parameters characterising the normal distribution has to be stored: the mean and the variance. The approximative distributions yields reasonably accurate gwc's and gammon probabilities compared to the exact one sided bearoff database. The size of the these approximative databases are roughly a quarter of the exact one, hence the limit is around the 13 to 15 point depending on the disk space available. The option to use approximative bearoff databases is work in progress!

As with the two sided bearoff databases it can be a rather time consuming task to generate one sided databases. The 10 point database takes approximately 2 hours to generate on the author's 1 GHz Pentium III. The 12 point database may more than one day!

17.1.3 Hypergammon databases

`gnubg` can also play Hypergammon; a variant of backgammon with only 3 chequers, but with counting gammons and backgammons. The starting position is

```
  GNU Backgammon  Position ID: AACgAgAAKgAAAA
                  Match ID   : cAkAAAAAAAAA
  +13-14-15-16-17-18------19-20-21-22-23-24-+     O: gnubg
  |                  |   |          X  X  X |     0 points
  |                  |   |                  |
  |                  |   |                  |
  |                  |   |                  |
  |                  |   |                  |
 v|                  |BAR|                  |     (Cube: 1)
  |                  |   |                  |
  |                  |   |                  |
  |                  |   |                  |
  |                  |   |                  |     On roll
  |                  |   |          O  O  O |     0 points
  +12-11-10--9--8--7-------6--5--4--3--2--1-+     X: jth
 Pip counts: O 69, X 69
```

gnubg can also play the two-chequer and one-chequer variations as well. Although it looks very simple to play, Hypergammon is in fact quite difficult to play correctly.

There are 3,276 possible ways to distribute 3 (or less) chequers over 25 points (the 24 points plus the bar). This gives a total of 10,732,176 possible two sided positions. However, many these are illegal:

Position type	**Number of positions**
Game over	6,551
Race (no contact)	587,926
Contact	7,365,427
Illegal positions	2,772,272
Total	10,732,176

So there is close to 8 million legal positions for Hypergammon. Since this is a relative small number it's possible to tabulate the game winning chance, cubeless equity, or cubeful equities for all positions.

Due to the contact positions it is not possible to generate such a database in one run; instead the process is iterative: (a) "guess" equities for the 8 million positions, (b) use these equities to calculate 1-ply equities for all positions, and (c) if the equities change significantly then go back to step (c).

17.2 Bearoff databases with GNU Backgammon

gnubg works with both one sided and two sided bearoff databases. Currently, it works with up to four databases; two of each kind. Two of the databases are read into memory for fast access, but of course these are limited to very small databases.

The one sided database kept in memory is either built-in or stored on disk as 'gnubg_os0.bd' depending on the compile time option USE_BUILTIN_BEAROFF, which is most easily controlled by passing the option --enable-bearoff=TYPE where type is either no, static, or external. gnubg can also work without this database, but will generate a heuristic version every time the program starts. This can take up to several minutes, depending on the speed of your computer.

We recommend using an external file as the executable will be smallest possible, and several running instances of gnubg will share the memory if your system has mmap (memory mapping of files).

The standard size is 15 chequers on 6 points, but you may generate a larger one, but remember that the gnubg will read the file into memory. The default size should be adequate for all purposes.

The two sided database kept in memory ('gnubg_ts0.bd') is optional and you may generate or download your own (see next section for details). gnubg will use the exact cubeful equities in the databases for cubeful money game evaluations and for truncation in money game rollouts (see (insert ref to rollout chapter)).

gnubg cannot directly use the equities for match play as the equities, in general, depend on the match score, cube value, and cube ownership. However, it's possible to calculate the cube efficiency parameter X from Rick Janowski's formulae from the cubeful money equities that are available. This is very important as the cube efficiency parameter often is very different from the range 0.6 and 0.68 that gnubg uses for long bearoffs and contact play, respectively. The calculated value of X is used in match play generalisation of Janowski's to yield cubeful match winning chances. gnubg uses a similar heuristic for Hypergammon (see below).

The two databases ('gnubg_os.bd' and 'gnubg_ts.bd') are kept on disk and accessed when needed. See next section for details on how to generate or download your own.

A few words of caution: the race neural net in gnubg is believed to be quite good, so it may not be worthwhile to generate large one sided database. Joseph Heled has published a comparison of chequer play decisions between the 0.13 race net and a one sided bearoff database for 15 chequers on 12 points. A total of 34,165 positions were examined, and the error rates were quite small:

Ply	Error rate/positions	Number of errors	Percentage of positions with errors
0	0.00025	3231	9.5
1	0.00032	3418	10.0
2	0.00002	1255	3.7

In order to play or analyse Hypergammon matches or session you need the databases 'hyper1.bd', 'hyper2.bd', and 'hyper3.bd' for 1-chequer, 2-chequer, and 3-chequer (normal) Hypergammon. These databases contain the cubeless game winning chance, gammon, and backgammon probabilities (based on cubeless money play without Jacoby rule), and cubeful equities for money game: centered cube (with Jacoby rule), centered cube (without Jacoby rule), owned cube, and unavailable cube. For match play gnubg uses a similar heuristic to the usage of the two-sided bearoff database in match play for normal backgammon: the cube efficiency parameter X from Janowski's formulae are estimated from the cubeful money equities.

Each position requires 28 bytes of storage. For easy indexing of the file gnubg also use storage for the illegal positions. The table below shows the size of the Hypergammon databases.

Number of chequers	Number of positions	Size of file/bytes
1	676	19,968
2	123,201	3,449,668
3	10,732,176	300,500,968

17.3 Obtaining bearoff databases

17.3.1 Generate your own

gnubg is supplied with a program 'makebearoff' that is used to generate bearoff databases. Due to various limitations it is only possible to generate bearoff databases with a size less than 2GB, i.e., the 13pt one-sided database and the 11 chequer two-sided database are the largest databases that can be generated with 'makebearoff'.

To generate one sided database issue

```
makebearoff -o 10 -f gnubg_os.bd
```

to generate the one sided 10 point database. The program 'makebearoff' uses a cache to store previously generated positions. You may set the cache size with the -s size option, e.g.,

```
makebearoff -o 10 -s 123456789 -f gnubg_os.bd
```

to use 123,456,789 bytes of memory for cache size. In general, if the cache size is less than the size of the database to be generated, then extra time is needed to generate the bearoff database because some positions may have to be calculated several times.

'makebearoff' can also reuse previously generated databases, so if you already had generated the 9 point database you can reuse it:

```
mv gnubg_os.bd gnubg_os9.bd
makebearoff -o 10 -O gnubg_os9.bd -f gnubg_os.bd
```

Note that 'makebearoff' requires temporary disk space to generate both one sided and two sided databases. In general, twice the disk space of the database to be generated is needed.

To generate a two sided database issue

```
makebearoff -t 6x8 -f gnubg_ts.bd
```

This example will generate the 8 chequers on 6 points database. Again, it's possible to adjust the amount of memory with the -s option. It's recommended to set the cache size to the maximum amount of memory available (although there is no need to set it beyond the size of the bearoff database to be generated).

Other options for 'makebearoff' are available, see makebearoff --help for the complete set.

At compile time the built-in one sided bearoff database is generated with the command

```
makebearoff -o 6 | makebearoff1 -o br1.c
```

which generated a 6 point one sided bearoff database. The database is piped into makebearoff1 that generates the C code for the database. The resulting 'br1.c' is more than 8MB, so it's not distributed with the source for gnubg.

The accompanying program 'makehyper' is used to generate databases for Hypergammon. For example, to generate the 3-chequer database issue the command

```
makehyper -c 3 -f hyper3.bd
```

Since the generation is *very* time consuming 'makehyper' will generate a checkpoint file (in the example above: 'hyper1.bd.tmp') that can be used to restart the calculation if needed by using the -r option. You can also change the default convergence threshold of 0.00001 if you're happy with less accurate equities. To generate the 3 chequer database you need approximately 400 MB of free memory. On a 2.4 GHz box with 512 MB of RAM the calculation ran for 58 iterations of 3000 seconds each, i.e., a total of 48 hours!

See makehyper --help for the complete set of available options.

17.3.2 Download

You may download the two sided database with 6 chequers on 6 points from ftp://alpha.gnu.org/gnu/gnubg/gnubg_ts0.bd.gz and the one sided database with 15 chequers 6 points from ftp://alpha.gnu.org/gnu/gnubg/gnubg_os0.bd.gz.

Later, other databases may be available for download or it may be possible to purchase these on CDROM or DVD for a nominal fee.

17.3.3 Verification of databases

To verify that your generated or downloaded bearoff database is correct, the table below lists the MD5 checksums for a number of possible databases.

17.3.3.1 One-sided bearoff databases (compressed)

The table below contains the MD5 checksums for the compressed one sided bearoff databases, i.e., databases generated with default options.

Chequers	**Points**	**MD5 checksum**
15	1	`c789f049ec98ce4e307b471257999f39`
15	2	`b6e61c1625ae4b3b164e93ab064192b9`
15	3	`346dae6139ccb4b227c534373e5c52e4`
15	4	`653255f5f9f22fd50277f7ff25b2a343`
15	5	`2064f9a56b23117d053a573c96a92fa2`
15	6	`3dc7b833c4670849cee00479a9e21b49`
15	7	`67235c8e0ee152df5daf36cbeae5b3c2`
15	8	`a4acbb5c7e9e1f08e561afe0af934e5c`
15	9	`9c4ddab4e51c3e668c9c97b8f8768dbc`
15	10	`81b3898f06bbd08ee8295a839251a10a`
15	11	`78ecb4be86dab6af8755ea4063d50fb6`
15	12	`770fcff48894a96ebb2249343ef94866`
15	13	`cc74b69a62f6e648936f533838a527a8`
15	14	not available
15	15	not available
15	16	not available
15	17	not available
15	18	not available

17.3.4 Two-sided bearoff databases (with cubeless and cubeful equities)

The table below contains the MD5 checksums for the default two sided bearoff databases.

Chequers	**Points**	**MD5 checksum**
1	6	`7ed6f8e7fce16ea2b80e07a4a516653c`
2	6	`e9d760bf213841c285245ed757a52f4d`
3	6	`9d67da3db32ad4720cc38eecf9a67967`
4	6	`9156f37032d1d4b0352a41186e632dfc`
5	6	`0db19ab08feae1feb33ddbd709479f62`
6	6	`44b6040b49b46cb9dd2ce8caa947044d`
7	6	`9eb8b042d4d2ddf8d40e74a892745ad5`
8	6	`fcdbbc80b7ef84ddc81b839d0f26bed1`
9	6	`a11b2d410d51401143d05e73f9ffac15`
10	6	`12dc70c86f356d06bc96ee38dee40c62`
11	6	not available
12	6	not available
13	6	not available

14	6	not available
15	6	not available

17.3.4.1 Two-sided databases for Hypergammon

As the generation of the Hypergammon databases are an iterative process it's not possible to give MD5 checksum for these, as it depend heavily on your convergence threshold, the number of restarts, and rounding errors.

Below is a random position from the database. The equities and percentages in your own database should be very similiar (if not identical):

```
 GNU Backgammon  Position ID: ADAAAQAkIAAAAA
                 Match ID   : cAkAAAAAAAAA
 +13-14-15-16-17-18------19-20-21-22-23-24-+     O: gnubg
 |       X     X    |   |                X |     0 points
 |                  |   |                  |
 |                  |   |                  |
 |                  |   |                  |
 |                  |   |                  |
v|                  |BAR|                  |     (Cube: 1)
 |                  |   |                  |
 |                  |   |                  |
 |                  |   |                  |
 | O                |   |                  |     On roll
 | O                |   |             O    |     0 points
 +12-11-10--9--8--7-------6--5--4--3--2--1-+     X: jth

             Player       Opponent
Position         3018          2831

Owned cube                  : -0.0230
Centered cube               : -0.2310
Centered cube (Jacoby rule) : -0.2186
Opponent owns cube          : -0.3548

        Win     W(g)    W(bg)   L(g)    L(bg)   Equity  (cubeful)
static: 0.456   0.244   0.014   0.318   0.019   (-0.168  (-0.219))


No double           : -0.219
Double, pass        : +1.000   (+1.219)
Double, take        : -0.710   (-0.491)

Correct cube action: No double, beaver
```

18 Absolute ratings

Kees van den Doel (`mailto:kvandoel@xs4all.nl`)

18.0.1 Summary

Simulations were performed to establish the relation between the rating difference (**R**) between backgammon players of various strengths, and the error rates for the two players. This will allow a match analysis to translate the error rate of each player into an estimation of the absolute ratings of the players, once the rating of "perfect play" has been defined. Of course, this estimate will be only as good as the quality of the analysis and will become invalid for the estimation of the rating of players at a comparable level as at which the analysis was performed. However, the measured relation does not depend on the analysis level and remains valid for very high levels of analysis such as rollouts.

Two sets of simulations were performed. In the first the relation between the difference in normalized error rate per unforced decision (**DEPD**) and rating difference **R** was measured. The main result of the first simulation set is an almost linear relation between **DEPD** and **R**, which depends somewhat on the match length. As a by-product of this investigation the relative strengths of various playing levels of GNUBG were determined precisely by rating.

The second set of simulations attempts to improve on the linear model of the first set, by separating out the effects of cube errors and chequer play errors. A bilinear fit was made of **R** to both the normalized chequer play error per unforced move (**EPM**) and the cube error per unforced cube decision (**EPC**). The resulting bilinear fit is shown to improve the prediction of the rating difference, especially for matches with very poor cube handling compared to chequer play.

The scripts used for the simulations and a readme file explaining their use are available here (`http://www.cs.ubc.ca/~kvdoel/tmp/ratings/gnubgScripts.zip`).

The final result is the following formula for the rating difference:
R = a2(N)*EPM+b(N)*EPC,
where
a2(N) = 8798 + 25526/N,
and
b(N) = 863 - 519/N.

18.1 Part I: Rating and DEPD

18.1.1 Introduction

In order to estimate the rating difference between playing levels we played a series of 1000 matches of a given match length between various levels and estimated the rating difference by analyzing the results. From the analysis we obtain the normalized error rate per unforced decision and the "luck adjusted result" (**P**), which is an estimate of the match winning probability using variance reduction. These numbers were averaged over the 1000 matches and the variance was also estimated. The rating difference **R** and a 90% confidence interval was then estimated from the 90% confidence interval of **P** using the FIBS rating formula, **P = 1/(1 + 10^(R*sqrt(N)/2000))**, with **N** the match length.

Simulations were performed for match lengths of 1, 3, 5, 7, 11, and 23. For practical reasons it was necessary to perform the error and luck analysis at 0-ply (expert level of GNUBG) and consider only levels at the expert level (0-ply) and below (0-ply with noise addition). Equal noise was generated for chequer play and for cube decisions. The following levels (0-ply with given noise) were paired:

player 1	**player 2**
0	0.005
0	0.015
0	0.02
0	0.04
0	0.06
0.04	0.06

Level pairings

The assumption that the rating difference between players depends only on **DEPD** and not on the absolute level of play was verified implicitly with this choice as the last pairing has both players with noise addition. If the absolute **EPD** mattered the last pairing would give results inconsistent with the first five, which is not the case except for 1pt matches where as described in the results section.

With these settings, the simulation took about 100 hours on a 950Mhz Pentium III. Simulations were performed on Windows XP, using GAWK and sh scripts within the GNU ULTILS in combination with the no-GUI version of GNUBG (build 03 08 07). They should work equally well on LINUX.

Three AWK scripts are used to create GNUBG scripts to play a given number of matches, analyze them, and to collect statistics. Separate scripts were used in order to modularize the tasks so the simulation can be interrupted and continued without loss of data. The resulting match statistics were then processed trough a fourth AWK script which computes the average rating **R**, and the average **DEPD** over the set, as well as the 90% confidence intervals as estimated for a variance estimation. The levels of the players and the analysis level is set in the .gnubgautorc file, outside the scripts. Further analysis was then performed in MATLAB.

18.1.2 Results

The results of the simulations are given in the figure below for all match lengths.

More detailed results are given in the figures below for each specific match length. The measured data points are indicated by the circles. The red line is a piecewise linear interpolation through the data points. The black lines indicate the 90% confidence interval. The green line is a least square fit through the data points (minimizing the sum of squares of absolute rating differences)

For all match length except 1, the linear fit seems quite good and we can approximate the rating difference by **R= a(N)*DEPD**, where **N** is the match length. The function **N*a(N)** is plotted in the figure below, along with a linear least square fit.

The linear fit of **N*a(N)** gives the following rating formula:
R = (11971 + 23681/N) * DEPD.
In the figure below the formula is plotted along with the actual data points;

Rating versus **DEPD** approximated by **R = (11971 + 23681/N) * DEPD**. The dotted lines are the prediction, the solid lines are piecewise linear segments through the data.

The bump in the curve for the 1pt match around **DEPD=0.015** is caused by the data point from the **0.04 - 0.06** (intermediate - beginner). Apparently the approximation that the rating difference does not depend on the absolute values of the error rates is less accurate for 1pt matches.

Note that, for the purpose of measuring **DEPD** between simulated players, it is a reasonable approximation to analyze a match at expert level if both players are playing at expert level or worse. This is because we are only interested in the difference between the error rates. When a match is re-analyzed at World Class level (2-ply) the individual error rates increase, but the difference remains approximately constant. To verify this explicitly, we ran 2 sets of 40 matches between 0-ply expert level and 0-ply advanced level (.015 noise) and analyzed them both at 0-ply and at 2-ply. The results are given in the table below.

0-ply	**2-ply**
0.0075	
0.0068	
0.0080	
0.0072	

2-ply versus 0-ply **DEPD** testimates

We see that the error in the 0-ply estimation appears to be in the order of 5%. As an additional verification, a set of 100 3 pt matches was played between the World Class level (2-ply) and the expert level which was analyzed at 2-ply. It was found that the resulting **DEPD** and rating difference **R** was well predicted by the model. The data point is indicated by the 'X' on the figure for the 3pt match above and fits well on the curve.

18.1.3 Conclusions

The data is well approximated by a linear relation between rating difference and error per decision difference of the form **R=a(N)*DEPD** where **N** is the match length. An excellent approximation for **a(N)** is **a(N) = 23681/N + 11971**. The approximation is worst for **N=1**. The assumption that **R** does not depend on the absolute **EPD** but only on the difference was validated, except for **N=1**. The assumption that**DEPD** for matches between players at or below the GNUBG expert level can we approximated by a 0-ply analysis was verified by re-analyzing some matches at 2-ply with small effect on the **DEPD**. An explicit simulation of a series of 2-ply versus 0-ply matches resulted in an estimated **R** which fitted the model very well, supporting the assumption that the linear relation between **R** and **DEPD** is valid for any level of analysis.

18.2 Part II: Rating and EPM and EPC

18.2.1 Introduction

In the above we have assumed that the rating depends only on**DEPD = (totMoveErrors + totCubeErrors)/totalDecisions**. In reality the rating difference will depend on some combination of chequerplay skill and cube skill, at which the above formula makes an educated guess. A more powerful predictor taking into account the separate effect of cube errors and chequerplay errors can be obtained by simulating matches with independently varied cube and chequerplay noise and fitting the rating by a bilinear form **R= a2(N)*EPM +b(N)*EPC**, with **EPC = moverErr/unforcedMoves** and **EPC=cubeErr/unforcedCubeDecisions**.

For match lengths of 3, 5, 7, 11, and 23, 500 matches between GNUBG 0-ply and GNUBG 0-ply with 16 different noise settings were played. The chequer play noise settings were **0, 0.02, 0.04, 0.06** and the cube play noise settings were **0.05, 0.1, 0.4, 0.8**, 16 combinations in total. For each set of 500 matches the **EPM** and **EPC** were averaged. The rating difference was estimated as in Part I by using the luck adjusted result.

For each match length the 16 data points were fitted to **R = a2(N)*EPM +b(N)*EPC** with a least square method, minimizing the squares of the rating differences. For **N=1** there are no cube decisions and **a2(1) = a(1) = 37916**.

18.2.2 Results

In the figures, below, we plot **R** against **DEPD**. The green "O" indicates the measure rating differences, the green dots the 90% confidence interval of the data, the red "X" indicates the linear fit using the formula from Part I, and the blue "X" indicates the prediction from the bilinear fit with the coefficients a2 and b as indicated.

The coefficients **a2(N)** and **b(N)** as a function of match length are well approximated by a linear function of **1/N**. We show these coefficients and a linear fit in the two figures below.

18.2.3 Conclusions

The bilinear fit improves the linear fit especially for large cube errors. Usually the linear estimate is too high, reflecting too much weight given to the cube errors, for which the bilinear formula corrects. The data is well approximated by
R = a2(N)*EPM+b(N)*EPC,
where
a2(N) = 8798 + 25526/N,
and
b(N) = 863 - 519/N.

18.2.4 Appendix: Misc. measurements

Player1	**Player2**	**Match length**	**R1-R2**	**DEPD (e2-e1)**	**EPM**	**EPC**
expInt	intExp	5pt	472 [451 493]	–	-0.0344	-0.01
expBeg	advExp	5pt	115 [103 127]	–	-0.00865	+0.0176
expExp	expBeg	5pt	31 [21 41]	–	0	-0.0185

expExp	expBeg	11pt	29 [23 36]	–	0	-0.0165
expExp	expBeg2 (1.0 cubenoise)	5pt	191 [176 206]	–	0	-0.192
wclassWclass	wclassExp	5pt	6 [-15 27]	–	–	–
1-ply	0-ply	5pt	13 [-4 30]	–	–	–

Some misc. measurements. expBeg for example means chequer play at level expert, cubeplay at level beginner. Rating interval is the 90% confidence interval.

19 Additional documentation

A tutorial written by Albert Silver is available at Tom Keith's Backgammon Galore All About GNU (`http://www.bkgm.com/gnu/AllAboutGNU.html`). An Italian translation is available from Tutto su GNU (`http://members.xoom.virgilio.it/free_bgammon/`) by courtesy of Renzo Campagna, Giulio De Marco and Alberta di Silvio.

GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled “History”, and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled “Acknowledgements" or "Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document
> under the terms of the GNU Free Documentation License, Version 1.1
> or any later version published by the Free Software Foundation;
> with the Invariant Sections being LIST THEIR TITLES, with the
> Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
> A copy of the license is included in the section entitled "GNU
> Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Command Index

E

S

Index

(Index is nonexistent)