The Gamebooklib Package*

Robert J Lee
latex@rjlee.homelinux.org

July 30, 2023

Abstract

This package provides macros and environments to allow the user to typeset a series of cross-referenced, numbered "entries", shuffled into random order.

1 Introduction

This package was written to allow the typesetting of gamebooks.

A gamebook is a book divided into "entries" (which may be a single paragraph or longer), each with a sequentially numbered value. At the end of each entry, a link to one or more other numbered entries is given; the reader selects one and they follow through the story in this order. Gamebooks traditionally start at the entry numbered "1" and some gamebooks have multiple endings, with the final section representing a success or victory for the reader.

In particular, this package handles two technical challenges which are tricky to solve with LaTeX directly: writing the "entries" in order but presenting them in a randomised order; and presenting footnotes at the end of each "entry", numbered in the order in which they appear, not the order in which they are written.

Aside from shuffling the paragraphs and fixing the handling of footnotes, this package provides little help for the actual typesetting of gamebook entries. For this, please consider using André Miede's `gamebook` package, available on CTAN[1]. That package is orthogonal to this one; both may be used together.

Terminology

For the purposes of this document, the term *entry* means a numbered section of text. This term helps describe the common format of a gamebook, while avoiding confusion with the terms *paragraph* and a *section*, both of which already have a clear definition.

The term *gamebook* means a book consisting of numbered sections, typically presented in a non-linear order with a tree or graph of possible reading options.

*This document corresponds to Gamebooklib Gamebook, dated 2023/07/28.

[1] `https://www.ctan.org/pkg/gamebook`

Usage

To load the class, use `\usepackage[`*options*`]{gamebook}`

The class options are described below; `[footnotes,jukebox,mark]` is normally a good choice, although there are some limitations.

The user simply writes entries like this inside the document:

```
\begin{gentry}{codea}
This is the first entry

Turn to \turnto{codeb}
\end{gentry}

\begin{gentry}[123]{codeb}[title]
This is the second and final entry.\par
To play again, turn to \turnto{codea}
\end{gentry}

% Output:
\thegentries
```

Here, two entries are defined, and then output.

The first entry is given a label of `gentry:codea`, and the `\turnto{codea}` in the second entry generates a link back to that label. You might define `\turnto` like this:

```
\newcommand{\turnto}[1]{\ref{gentry:#1}}
```

As `entry` is a common term likely to conflict with other environments, the environment to declare an entry has been named `gentry` (gamebook entry) instead.

Sometimes it is convenient, when writing an entry, to give it a fixed index in the text. The first optional argument in the second entry — `[123]` — fixes the entry at that position during shuffling. This is implemented by iterating over each entry and swapping it with that index. Duplicates are not currently checked for, but they would not be lost; if two entries are defined with the same fixed index, one will end up somewhere else in the text.

The first and last entries are automatically fixed without needing to specify the optional number.

Specifying the optional number as a value greater than or equal to the number of entries, or less than 2, is not recommended. (Yes, this example used `123`, which is beyond the number of entries. This was done to show the syntax).

The second optional argument is a title to be displayed alongside the numerical value.

Finally, the command `\thegentries` will output all entries defined so far, handling shuffling and footnotes.

`\gentryheader` `{`⟨*counterIdx*⟩`}{`⟨*fixedIdx*⟩`}{`⟨*code*⟩`}{`⟨*title*⟩`}`

Users may also want to redefine `\gentryheader` to change how the entries are displayed. The initial version is quite basic and intended for debugging; users should use `\renewcommand` to change it to something more pleasant.

Here's one suggestion, using packages from CTAN. It works well if you have either a title, or at least 2 lines of text before the first paragraph break inside the `gentry`.

It also requires near the start of the document:

```
\usepackage{lettrine}
\usepackage{etoolbox}
\usepackage{color}
```

And this anywhere before `\thegentries`:

```
\renewcommand{\LettrineFontHook}{%
  \color[gray]{0.5}\fontfamily{ppl}\fontseries{bx}}
\renewcommand{\gentryheader}[4]{%
 % \typeout{Typesetting entry at original index #1}
 \lettrine[lines=3,lhang=0.2,loversize=0.2]{\raisebox{0.1em}{\arabic{gentryctr}}}%
     {\textbf{\Large\textbf{\raisebox{0.3em}{~#4}%
     }}}%
     \ifstrempty{#1}{}{\linebreak\mbox{}~}}%
```

On the last line, `\mbox` stops the `\parindent` space being ignored; the ~ brings it back up to the expected spacing. You may need to adjust the `lettrine` spacing parameters, depending on the font size and whether you use header text.

Package options

These options can be specified as a comma-separated list to the `\usepackage` line.

`footnote`

The `footnote` option causes LaTeX to output footnotes at the end of each entry, or the end of each page, whichever comes first after the footnote mark.

There are some limitations, described below.

Support for other packages that affect footnotes is limited. If using `hyperref`, it is recommended to pass `[hyperfootnotes=false]` to avoid broken links.

`jukebox`

Use the Jukebox Index shuffling algorithm[2]. This is slightly slower, but tends to reduce the number of times you get a "turn to " instrution referencing the next (or previous) entry in the original order, by modifying the shuffle to ensure that adjacent gentries in the input have much less chance of being adjacent in the final document. `jukebox` requires a LaTeX 2_ε that supports `\numexpr`, and a minimum of 6 gentries. If fewer than 6 `gentry` environments are supplied before `\thegentries`, this option will only log a warning in verbose mode.

`noshuffle`

The `jukebox` option guarantees no more adjacent entries than without the option, for a given `seed` value; it may not eliminate them completely unless the number of entries is large. The performance is linear to the number of entries.

In general, it's easier to write gamebooks in a more linear fashion, in which related entries are kept together. But this is much less fun to play, as it's too easy for the reader to simply read the adjacent eentries to decide what to do. For this reason, this package shuffles the output entries by default. The first and last entries are never shuffled.

[2] `https://github.com/robertjlee/jukeboxshuffle`

Sometimes, perhaps for proofreading, you may not want the entries to be shuffled. In this case, you can use the `noshuffle` option.

`verbose` This macro causes the package to output information messages about what it's doing to the log file. This is not generally too useful, but it does include a mapping of the original paragraph indexes to their sorted positons, which may be useful to keep for handling proofreading corrections.

`endpage` Puts the last entry on a page on its own. This typically produces a better "winning" feel for reaching the last entry, but it can also produce documents with ugly spacing, so it's recommended to try it each way and see which works better for your text.

`seed` It's suggested that users specify a value for "seed" for stable builds, by adding the following before including this package: `\usepackage[seed=123]{lcg}`

Footnotes

When typesetting a gamebook with footnotes, it is confusing if they migrate to the bottom of each page, as the footnote becomes visually detached from the entry to which it relates. Some effort has been taken to ensure that footnotes can be typeset at the bottom of the page on which the mark appears, or the bottom of each entry, whichever comes first.

However, this implementation has some limitations:

- Footnotes are not expanded until they are typeset.
- As a consequence, attaching one footnote to another with the use of `\footnotemark` within a `\footnote` argument will not advance the counter in time, so `\footnotetext` may not behave as you expect unless it is also set inside the first footnote's text.
- Footnotes at the end of the page will not be broken across pages. Putting sufficient text into footnotes will cause the page to overflow.
- Where the footnote **mark** appears at the very end of a page, the footnote **text** may be set at the top of the subsequent page. TEX is asked not to break the page there, but this influence is limited. It may be possible for the author to avoid this, eg by adding a rubber length to the interline spacing (it may be easier to simply choose a different `seed` value for the `lcg` package to reshuffle).
- If you have a lot of rubber space in the text, or variably-sized items, LATEX may expand the footnote at the end of the entry before it works out where to put the page break. In this case, the footnote will appear on the wrong page. The macro `\noentryfoot` is provided to allow the author to fix this case.
- Support for other footnotes packages may be limited and the behaviour of packages like `footnote`, `endnotes`, `footmisx`, `fnote`, `dblfnote` *etc* may be affected.
- If using the `hyperref` package, it is recommended to pass the option `hyperfootnotes=false` to that package, as the footnote links will be incorrect.

- The package uses TEX `\mark`s to indicate which footnotes should appear on each page. Because TEX supports only one set of marks, this would break any other package or usage of `\mark` while a gamebook was being output.
- Because `\mark` does not escape a floating environment, such as `minipage`, it is likely that this footnote implementation will not work as expected if the gamebook or `\footnote` is set in a `minipage` or similar.
- Each footnote is evaluated in a group. Just conceivably, this might affect the behaviour of some macros that affect the document beyond the footnote.
- We rely on some non-"public" macros, such as `\@mpfn` and `\@footnotetext` defined by LATEX standard document classes. Other document classes may override these, which would override this package's footnotes too.

For this reason, the improved footnote handling is only enabled if you pass the `footnote` option, and only while the environment is active.

Implementation

```
\ProvidesPackage{gamebooklib}[2023/07/28 Gamebook by R Lee latex@rjlee.homelinux.org]
```

We need LATEX 2_ε, for the extra token registers.

verbose

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
```

The package option `verbose` enables detailed logging. Logging is via the macro `\gamebook@info`, which throws away detail messages unless the `verbose` option is given.

```
\newcommand{\gamebook@info}[1]{}%
\DeclareOption{verbose}{%
  \renewcommand{\gamebook@info}[1]{\PackageInfo{gamebooklib}{#1}}%
  \gamebook@info{Gamebook Library package is logging}}%
```

mark

RL: Untested start of mark support The `mark` package option enables support for LATEX 2_εMark interface, intruduced in 2022. This enables `gamebooklib` to declare its own `\mark` class, ensuring that other packages' marks are unaffected.

```
%% \newcommand{\gamebooklibmarkclass}{gamebooklibmark}
\def\gamebooklib@mark{\mark}
%% \DeclareOption{mark}{%
%%   \gamebook@info{Using mark class \gamebooklibmarkclass}%
%%   \NewMarkClass{\gamebooklibmarkclass}%
%%   \renewcommand{\gamebooklib@mark}[1]{\InsertMark{\gamebooklibmarkclass}{#1}}%
%% }%
```

endpage

The `endpage` option puts the last entry on its own page. This can work better when the last entry is about a page long, and also the final "winning" entry of the gamebook.

```
\newcommand\gamebook@beforelast{}
\DeclareOption{endpage}{%
  \renewcommand\gamebook@beforelast{\eject}%
}%
```

jukebox

The `jukebox` option defines the `\gamebox@jukebox` macro; while the macro does nothing, `\ifcsname` can then be used to determine if the option was set.

```
\DeclareOption{jukebox}{%
  \newcommand\gamebook@jukebox{}%
```

```
  \gamebook@info{Gamebook Library to perform jukebox index reshuffle
    pass}%
}%
```

footnote

The `footnote` option enables our footnote processing, to throw out footnotes at the end of each `gentry` so that they don't appear to be against subsequent entries for that page.

This is genearlly recommended, unless you have a reason to turn it off (such as a conflicting package). It's disabled by default because it could cause unexpected faults.

```
\def\if@gamebook@footnotes{\iffalse}
\DeclareOption{footnote}{%
  \gdef\if@gamebook@footnotes{\iftrue}%
  \gamebook@info{Gamebook Library footnotes per gamebook entry}%
}%
```

noshuffle

```
\def\if@gamebook@shuffle{\iftrue}
\DeclareOption{noshuffle}{%
  \gdef\if@gamebook@shuffle{\iffalse}%
  \gamebook@info{Gamebook Library entries output in order}%
}%
```

seed

All unknown options are passed to `lcg`, as it's our only dependency with options.

```
\DeclareOption*{%
  \PassOptionsToClass{\CurrentOption}{lcg}%
}%
\ProcessOptions\relax%
```

We need to capture environment contents

```
\RequirePackage{environ}%
```

`Macroswap`: used to swap the commands that evaluate the macros

```
\RequirePackage{macroswap}%
```

`Ifthen` makes branching and loops a little easier

```
\RequirePackage{ifthen}
```

`LCG`: random numbers for the shuffle

```
\RequirePackage{lcg}%
```

`Silence` is used to suppress a warning from `lcg` that gamebook is **not** wasting counter registers. `debrief` ensures that the user is at least told than warnings were suppressed.

To see the warnings, put the following line before `\usepackage{gamebook}`: `\usepackage[debrief,showwarnings]{silence}`:

```
\RequirePackage[debrief]{silence}%
```

Let's count the entries we're reading in so we can build up token registers.

```
\newcounter{gentryctr}%
\setcounter{gentryctr}{0}%
```

gentry

`[`⟨*fixedIdx*⟩`]{`⟨*code*⟩`}[`⟨*title*⟩`]` First, we need to parse the optional arguments.

When we say `\begin{`gentry`}`, LaTeX does some stuff ending in `\gentry` so we redefine `\gentry` to take an optional argument and delegate to `\@gentry`

```
\newcommand{\gentry}[1][]{\@gentry{#1}}%
```

`\gentry` just makes the first argument mandatory.

NB: If you define two entries with the same code, LaTeX will print out a "multiply defined" label warning.

```
\newenvironment{@gentry}[2]{%
  \xdef\gentryidx{#1}%
  \xdef\gentrycode{#2}%
  \@@gentry%
}{\ignorespacesandallpars%
  \global\let\gentryidx\@undefined%
  \global\let\gentrycode\@undefined%
  \global\let\gentryidxu\@undefined%
  \global\let\gentryidxs\@undefined%
}%
```

`gentryidx` This can be used inside a `gentry`; it expands to the first optional argument of the `gentry` environment, which is either blank or the requested fixed index of the entry. To get the actual shuffled index, use `\gentryidxu`.

`gentrycode` This can be used inside a `gentry`; it expands to the first mandatory argument of the `gentry` environment, which is the code for this entry (without the `gentry:` prefix).

`\@@gentry` then reads in the optional title argument, storing it in the `\gentrytitle` macro to supply the unsorted index number and the current entry's code respectively.

```
\newcommand{\@@gentry}[1][]{%
  \def\gentrytitle{#1}%
  \stepcounter{gentryctr}%
```

For fixed entries, define a macro to hold the requested index

```
  \ifthenelse{\equal{\gentryidx}{}}{}{%
    \expandafter\xdef\csname fixedat\arabic{gentryctr}%
    \endcsname{\gentryidx}%
  }%
  \expandafter\global\expandafter\newtoks\expandafter{%
    \csname paratok\arabic{gentryctr}\endcsname{ }%
  \Collect@Body\gentry@store%
```

This was supposed to discard any blank lines at the end of the `gentry` environment, but it still left odd spacing in the output somehow, and sometimes produced weird error messages about `\inaccessible` and `\head`. Removed for now.

```
% \ignorespacesandallpars%
}
```

Store the collected environment contents for `\thegentries` to output: This uses the token register `\gentry`N, where N is the unsorted index of this entry. Later, we'll change the values of the N bit of `\gentry`N macros when we shuffle (the underlying token registers stay the same).

This relies on `\refstepcounter{`⟨*gentryctr*⟩`}` being expanded before this macro. The label `gentry:\gentrycode` is thus set to the current index of the final output entry.

```
\newcommand{\gentry@store}[1]{%
  \edef\head{\noexpand\begingroup\noexpand\gentryheader%
    {\arabic{gentryctr}}{\gentryidx}{\gentrycode}{\gentrytitle}%
    \noexpand\label{gentry:\gentrycode}%
  }%
  \global\expandafter\csname paratok\arabic{gentryctr}\endcsname=%
```

Output the header, the environment token list, flush any footnotes (if applicable) then the inter-gentry footer.

```
  \expandafter{\head #1%
    \outputfootnotes@endgentry%
    \gentryfooter\endgroup}%
}%
```

`\gentry@footnotespergentry` This macro is executed inside the `gentry` group and sets up the commands to be run to allow footnotes. It is only expanded if `\if@gamebook@footnotes` is true, *ie* the `footnotes` option is given.

```
\newcommand{\gentry@footnotespergentry}{}
```

`\thegentries`

This macro shuffles the entries as required, then expands to them in the correct order.

```
\newcommand{\thegentries}{%
```

This command is in a group so that the output routine resets.

The expansion of `\gentry@footnotespergentry` only happens if the `footnote` option was given; it sets up the output routine while the gamebook is running.

```
\begingroup%
  \if@gamebook@footnotes\gentry@footnotespergentry\fi%
```

To begin, let's record the number of entries. This may come in useful. Note that you can use this macro in the `gentry` environment, because that's not been expanded yet.

```
  \xdef\gentrycount{\arabic{gentryctr}}%
```

The next thing is to perform some surgery on LCG. This cuts out an **annoying** warning, hopefully more reliably than replacing the definition of `\p@stkeysr@nd`. The warning is simply that this package doesn't waste another counter every time it changes the random limits (which happens a lot during the Fisher-Yates shuffle):

```
  \WarningFilter{lcg}{Using an already existing counter rand}%
```

The output routine and end-of-page footnotes

The idea is to keep footnotes always on the same page as their mark where possible.

LaTeX does lots of fun things with the output routine, which we want to keep. So grab a copy of whatever the code is currently doing:

Here I'm using the `\edef` trick to expand `\the\output` into a token register, because using a macro causes a weird error about an "`{`" after "`\the`".

```
\newtoks\gentry@oldoutput{}%
\edef\mytmp@{\noexpand\gentry@oldoutput={\the\output}}\mytmp@%
```

For the footnotes, if we reach the end of a page without outputting them, we need to flush them.

`\output` is the output routine. It takes the page built up in `\box255`, annotates it with headers, footers etc, then ships it out. For our purposes, we only need to append the footnotes to the bottom of `\box255`.

```
\if@gamebook@footnotes\output={%
  \def\gentry@deferoutput{\the\gentry@oldoutput}%
```

The `\outputpenalty` tells us why the output routine was called; generally, it's invoked whenever a new floatable environment is generated, or when a page is full. Anything less than -1000 means that the page was filled, so we should add any footnotes only in this case.

```
  \ifnum\outputpenalty<-\@M\else%
    \if\gentryshouldoutput0%
      \unvbox255\def\gentry@deferoutput{}%
    \else%
      \expandafter\ifcsname footnotetoks\botmark\endcsname%
      \expandafter\if\expandafter\relax\expandafter%
      \detokenize\expandafter{\csname footnotetoks\botmark\endcsname}\relax\else%
      \global\setbox255=\vbox to \vsize{%
        \unvbox255\vfill\outputfootnotes@endpage}%
    \fi\fi%
  \fi\fi%
  \gentry@deferoutput%
%  \the\gentry@oldoutput
}\fi%
```

The Shuffling Algorithm

The basic shuffling algorithm is to first shuffle all entries, except for those marked with a fixed index, then to go through the fixed-index entries in order and swap them into their final place.

The original version of this package had a bug relating to multiple fixed-index entries (now fixed). In short, let A, B, and C be indices; if $A < B$ and (unshuffled) entry number A was fixed at (shuffled) location B, while (unshuffled) entry number B was fixed at (shuffled) location C, so during the "shuffle," A would be swapped with B, then B would be swapped with C, resulting in A appearing at C in the text, not B as requested. Because the unshuffled index doesn't appear in either the source or output document, this could be difficult to diagnose; the author simply saw one of their entries ending up in the wrong place.

TEX has well beyond 255 token registers these days, so don't bother to check that limit.

The `LCG` package provides a suitable pseudo-random number generator. What we want is a repeatable series of disparate numbers, not an especially random one.

1. Work out how many entries there are (N). Provided `\thegentries` is called at the end, this is just the value of `gentryctr`
2. Declare a set of token registers named `\paratoks`n, where n is each integer $1 \ldots N$ inclusive. These will hold the contents of the entry.
3. Declare a set of macros named `\paraidx`n, where n is each integer $1 \ldots N$ inclusive, each initialised to n. These will hold the number of the entry.
4. Shuffle elements $\{2 : N-1\}$, in that array. For $i = 2$ through $N-2$
   (a) Let R be a random number between i and $N-1$ inclusive
   (b) If $R \neq i$ then swap macros `\gentrytoks`R and `\gentrytoks`i
   (c) If $R \neq i$ then swap macros `\paraIdx`R and `\gentryidx`i

5. If a jukebox index sort is requested, perform an optimisation pass (see below)

6. For $i = 1 : n$, output token reg i

Define macros `\csname paraIdx`n`\endcsname` containing the arabic original gentry number to be put out on the nth output gentry.

`gentrycount`

```
\xdef\gentrycount{\arabic{gentryctr}}%
```

we can reuse `gentryctr`; we've finished this set of paras and kept the total count.

```
\setcounter{gentryctr}{0}%
```

If there are fewer than 3 entries, don't try and shuffle.

```
\ifthenelse{\gentrycount<3}{}{%SHUFFLE START
\whiledo{\not{\value{gentryctr}>\gentrycount}}{%
  \edef\gentryidxu{\arabic{gentryctr}}%
  \expandafter\xdef\csname paraIdx\gentryidxu\endcsname{\gentryidxu}%
  \typeout{DEFINED paraIdx\gentryidxu}%
  \stepcounter{gentryctr}%
}%
```

`gentryidxu` The `\gentryidxu` macro can be used inside a `gentry` to obtain the current arabic shuffled index of the entry.

`gentryidxs` The `\gentryidxs` macro can be used inside a `gentry` to obtain the arabic original unshuffled index of the entry (the first `gentry` is 1, and this counter resets after each expansion of `\thegentries`).

```
\if@gamebook@shuffle%
\setcounter{rand}{\gentrycount}%
\addtocounter{rand}{-1}\edef%
\stoppoint{\arabic{rand}}%
```

First, shuffle everything that isn't fixed down. Don't renumber para 1 or `\gentrycount`; Fisher-Yates-shuffle the rest NB: we stop at `\gentrycount`-2, because `\gentrycount`-1 would only shuffle with itself.

```
\setcounter{gentryctr}{2}%
\chgrand[last=\stoppoint]%
\whiledo{\value{gentryctr}<\stoppoint}{%
```

If this is to be swapped with a fixed position, skip it

```
  \edef\gentryidxu{\arabic{gentryctr}}%
  \expandafter\ifcsname fixedat\gentryidxu\endcsname%
    \gamebook@info{Not shuffling \gentryidxu; fixed pos}%
    \stepcounter{gentryctr}%
  \else%
    \gamebook@info{Shuffling \gentryidxu}%
    \stepcounter{gentryctr}%
    \edef\nextidx{\arabic{gentryctr}}%
```

Roll the dice. If we've hit an entry with fixed position, we must skip it, or it would end up being swapped out into `fixedat\arabic{rand}` instead.

```
    \chgrand[first=\nextidx]%
    \rand%
    \expandafter\ifcsname fixedat\arabic{rand}\endcsname\else%
```

```
        \gamebook@info{Shuffling \gentryidxu to \arabic{rand}}%
        \macroswap{paraIdx\gentryidxu}{paraIdx\arabic{rand}}%
      \fi%
    \fi%
  }%
```

Now move fixed entries into their final place:

```
  \setcounter{gentryctr}{2}%
  \whiledo{\not{\value{gentryctr}>\stoppoint}}{%
    \edef\gentryidxu{\arabic{gentryctr}}%
    \expandafter\ifcsname fixedat\gentryidxu\endcsname%
      \expandafter\edef\expandafter\mydest\expandafter%
        {\expandafter\csname fixedat\gentryidxu\endcsname}%
        \gamebook@info{MOVING FIXED GAMEBOOK ENTRY INTO PLACE: \gentryidxu -> \mydest}%
        \macroswap{paraIdx\gentryidxu}%
                {paraIdx\expandafter\csname fixedat\gentryidxu\endcsname}%
```

Edge case: It's possible that we also have `fixedat\mydest`; in which case that would be messed up with the reshuffling. So we need to rename that to `fixedat\gentryidxu` as well, then reprocess this index

```
      \expandafter\ifcsname fixedat\mydest\endcsname%
        \macroswap{fixedat\gentryidxu}{fixedat\mydest}%
        \expandafter\global\expandafter\let\csname fixedat\mydest\endcsname\@undefined%
        \addtocounter{gentryctr}{-1}%
      \fi%
```

if we are doing a jukebox shuffle, remember which final entries are fixed, so they don't get moved.

```
      \ifcsname gamebook@jukebox\endcsname%
        \expandafter\def\csname fixedto\mydest\endcsname{}%
      \fi%
    \fi%
    \stepcounter{gentryctr}%
  }%
```

The jukebox shuffle requires an extra pass. This must come after moving fixed entries into their final place, to allow us to compare the initial indicies. We make a reasonable effort:

- t is the current index
- u is the next index
- r is a random index after u (make 3 attempts to find a non-fixed r)
- if $\text{abs}(t-u) = 1$ and r is not fixed, then swap u and r **if** u is not fixed; otherwise:
- if $\text{abs}(t-u) = 1$ and r is not fixed, then swap t and r **if** t is not fixed;
- otherwise, give up.

```
  \ifcsname gamebook@jukebox\endcsname%
  \ifnum\gentrycount<6%
    \gamebook@info{Jukebox pass skipped; too few entries}%
  \else%
  \setcounter{gentryctr}{2}%
```

```
\whiledo{\not{\value{gentryctr}=\stoppoint}}{%
  \edef\gentryidxt{\arabic{gentryctr}}%
  \edef\curpos{\csname paraIdx\gentryidxt\endcsname}%
  \stepcounter{gentryctr}%
  \edef\gentryidxu{\arabic{gentryctr}}%
  \edef\nextpos{\csname paraIdx\gentryidxu\endcsname}%
  \edef\pdiff{\the\numexpr\curpos+\nextpos}%
  \ifnum\pdiff<0\edef\pdiff{\the\numexpr-\pdiff}\fi%
  \edef\sdiff{\the\numexpr\curpos-\nextpos}%
  \ifnum\sdiff<0\edef\sdiff{\the\numexpr-\sdiff}\fi%
  \ifnum\pdiff<\sdiff\relax\def\thediff{\pdiff}\else\def\thediff{\sdiff}\fi%
  \ifnum\thediff=1%
    \gamebook@info{Jukebox: entries are too close: %
      \gentryidxt,\gentryidxu\space (original \curpos,\nextpos)}%
    \chgrand[first=\numexpr\gentryidxu+1]%
    \rand%
    \expandafter\ifcsname fixedto\arabic{rand}\endcsname\rand\fi%
    \expandafter\ifcsname fixedto\arabic{rand}\endcsname\rand\fi%
    \expandafter\ifcsname fixedto\arabic{rand}\endcsname%
      \gamebook@info{Can't reshuffle: failed to find non-fixed index}%
    \else%
      \ifcsname fixedto\gentryidxu\endcsname%
        \ifcsname fixedto\gentryidxt\endcsname%
        \gamebook@info{Can't reshuffle: \curpos\space and %
          \nextpos\space are both fixed.}%
        \else%
          \gamebook@info{Reshuffling \gentryidxt\space to \arabic{rand}}%
          \macroswap{paraIdx\gentryidxt}{paraIdx\arabic{rand}}%
        \fi%
      \else%
        \gamebook@info{Reshuffling \gentryidxu\space to%
          \arabic{rand} (alt)}%
        \macroswap{paraIdx\gentryidxu}{paraIdx\arabic{rand}}%
  \fi\fi\fi%
}%
\fi\fi%
}\fi%SHUFFLE END
```

Now we can output the `gentry` token registers to let LaTeX do its thing:

```
\gamebook@info{Shuffled! Gentry order:}%
\setcounter{gentryctr}{1}%
\whiledo{\not{\value{gentryctr}>\gentrycount}}{%
  \edef\gentryidxu{\arabic{gentryctr}}%
  \gamebook@info{\gentryidxu -> \csname paraIdx\gentryidxu\endcsname}%
  \stepcounter{gentryctr}%
}%
```

We can reuse the same counter again to output

```
\setcounter{gentryctr}{0}%
\gamebook@info{Outputting \gentrycount\space gamebook entries}%
\whiledo{\value{gentryctr}<\gentrycount}{%
```

Use refstepcounter in the loop to allow `\label` to work as expected.

```
  \refstepcounter{gentryctr}%
```

Keep last entry on its own page

```
    \ifthenelse{\value{gentryctr}=\gentrycount}{%
      \gamebook@beforelast%
    }{}%
    \edef\gentryidxu{\arabic{gentryctr}}%
    \xdef\gentryidxs{\csname paraIdx\gentryidxu\endcsname}%
    \gamebook@info{Output gentry \gentryidxu\ of \gentrycount,%
      original idx \gentryidxs}%
```

Output the stored entry body, stripping any extraneous space:

```
    \the\csname paratok\gentryidxs\endcsname%
  }%
```

Finally, we clear the registers and reset the counter in case we want to start again (NB: `fixedto` is scope to the current block only, so no need to clear that)

```
  \gamebook@info{All gamebook entries added to main vertical list}%
  \setcounter{gentryctr}{1}%
  \whiledo{\not{\value{gentryctr}>\gentrycount}}{%
    \edef\gentryidxu{\arabic{gentryctr}}%
    \expandafter\global\expandafter\let%
      \csname paratok\gentryidxu\endcsname\@undefined%
    \expandafter\ifcsname fixedat\gentryidxu\endcsname%
      \expandafter\global\expandafter\let%
        \csname fixedat\gentryidxu\endcsname\@undefined%
    \fi%
    \stepcounter{gentryctr}%
  }%
  \setcounter{gentryctr}{0}%
  \eject%
  \endgroup%
}%
```

The final `\eject` ensures that the output routine has flushed as many pages as it can, before the output routine is reset again.

`\gentryheader` `{`⟨*counterIdx*⟩`}{`⟨*fixedIdx*⟩`}{`⟨*code*⟩`}{`⟨*title*⟩`}`

This macro is called before outputting the header. Its job is to format whatever header information the user wants to see on each entry; generally, this will be the page number.

It takes 4 parameters:

1. The *unshuffled* index value (to help an author find an entry in the original text); this is numerically the same as `\gentryidxs`
2. The *fixed* index value, if any; otherwise an empty argument
3. The user-entered unique code for this entry
4. The user-supplied title, if any; otherwise an empty argument

The arabic value of the output index can be obtained with `\gentryidxu`; the numeric value is also set in the counter `gentryctr`.

```
\newcommand{\gentryheader}[4]{%
  \noindent\textbf{\Huge\arabic{gentryctr}\large\ #4}%
  \nopagebreak%
  \vspace{0.3em}%
  \nopagebreak%
  \par%
```

```
  \marginpar{#3}%
}%
```

`gentryshouldoutput` This macro **may** be called from the output routine, and can be used to suppress page breaks. It was added because it proves fairly easy to write custom divider routines that can produce blank pages. If it expands to the number 1, then the page will be output; if it expands to 0, it will not be.

For example, the following will prevent pages that are less than 80% full.

```
\renewcommand{\gentryshouldoutput}{%
   \ifdim\pagetotal>0.8\pagegoal\relax1\else 0\fi}
```

Caution: if this macro continually tests false, then material will eventually be discarded from the main vertical list to ensure that TEX can complete the output of the document. If you redefine this macro, make sure to check out output carefully for missing text.

```
\newcommand{\gentryshouldoutput}{1}
```

`gentryfooter` This takes no arguments and is simply expanded after the entry is typeset. The default adds some vertical space and a simple separator.

```
\newcommand{\gentryfooter}{%
  \par\vspace{2em}\centerline{---}\vspace{2em plus 1in}\par%
}%
```

`ignorespacesandallpars@` This is a technique described on StackExchange[3].

This is used to ensure that extra space at the start and end of the `gentry` environment is ignored.

```
\def\ignorespacesandallpars@{%
  \@ifnextchar\par%
    {\expandafter\ignorespacesandallpars\@gobble}%
    {}%
}%
```

`\@footnotetext` `{`⟨*token register*⟩`}` Token registers to hold each footnote.

Each footnote is held in its own token register, so that we can control which footnotes appear on each page.

LATEX would normally use saveboxes to store footnotes, but I prefer to hold off on expanding them until we know which page they're to be expanded on, which saves some difficulties (and perhaps creates others).

To start with, modify LATEX's `\@makefnmark` to output a mark for the footnote, holding its index. This tells the output routine which footnotes to include at the end of the page. The `\in@out` macro is only defined when outputting footnotes, so suppresses this mark when we don't need it.

```
\g@addto@macro{\@makefnmark}{%
  \ifcsname in@out\endcsname\else%
    \gamebooklib@mark{\arabic{\@mpfn}}%
  \fi%
}%
```

[3] https://tex.stackexchange.com/questions/179016/ignore-spaces-and-pars-after-an-environment#179034

The `\footnotetext@save` is a convenience that takes the the footnote counter value in the first argument, the token register as a second argument, and the token list (footnote text) as the third.

```
\g@addto@macro{\gentry@footnotespergentry}{%
\newcommand{\@footnotetext@save}[3]{%
  \global\newtoks#2{}%
  \global#2={\noindent\@footnotemark{}#3}%
}}%
```

`\@footnotetext@save` unpacks the arguments for `\@footnotetext@save@`. This one uses the value of the `\@mpfn` counter to unpack the footnote counter, the token register (full csname) as the first argument, and the token list (footnote text) as the second.

```
\g@addto@macro{\gentry@footnotespergentry}{%
\newcommand{\@footnotetext@save@}[2]{%
  \edef\@tmp@{\expandafter\arabic{\@mpfn}}%
  \expandafter\@footnotetext@save\expandafter{\@tmp@}{#1}{#2}%
}}%
```

Footnotes are built in a group, so that `\in@out` can be defined locally to indicate that footnotes are being built (which stops spurious `\mark`s).

```
\g@addto@macro{\gentry@footnotespergentry}{%
\renewcommand{\@footnotetext}[1]{%
  \begingroup%
  \def\in@out{}% flag that we're building footnotes
  \edef\@tmp{\expandafter\csname footnotetoks\arabic{\@mpfn}\endcsname}%
  \expandafter\@footnotetext@save@\expandafter{\@tmp}{%
    % This fixes up the use of \cs{footnotemark} within footnotes:
    #1}%
  \endgroup%
}}%
```

`\outputfootnotes` `{`⟨*maxIdx*⟩`}`

Command to output the footnotes, which are just in `\footnotetoks`N for now. An end user can call this at any point in the text to set out the footnotes.

This macro takes one argument, being the maximum index of footnotes to output, inclusive. Footnotes after this index will be excluded. If not provided, the value provided by the counter `\@mfpn`, which is the default LaTeX counter, will be used.

```
\newcounter{fncounter}%
\newcommand{\outputfootnotes}[1]{%
  \begingroup%
  \def\in@out{}% flag that footnotes are outputting; suppresses marks
  \setcounter{fncounter}{1}%
```

Called in vertical mode, and don't want to throw a page break.

I'm not sure how LaTeX renders the footnote rule exactly, so I'm just using my own.

```
  \def\footnote@rule{%
```

The next line comes from the TUGboat suggestions, and protect against various user changes

```
    \leftskip=0pt\rightskip=0pt\interlinepenalty=1000%
    \penalty-1000%
```

```
    \vspace{1pt plus 2pt minus 0.5pt}%
    \hspace{-0.5in}\rule{1.5in}{0.4pt}\\%
  }%
```

Invoke the output routine. This attempts to stop the output routine being invoked while we're adding the footnote rule, which could cause a blank footnote to appear.

Instead, it means some footnotes could appear on the subsequent page. To guard against this, footnotes are set into a `\vbox` to prevent page breaking.

The macro `\footnote@rule` is output before each footnote. The first time, it outputs a divider rule; subsequently, it just throws a new paragraph to keep footnotes on separate lines.

The next line comess from the TUGboat suggestions, and protect against various user changes

```
    \outputpenalty=-\@MM\break%
    \vbox{%
      \whiledo{\not{\value{fncounter}>#1}}{%
```

We are now looping over all possible entry numbers, in order. Some will already have been output, but we check them all anyway (it doesn't take much time).

First, we check if the macro exists. If it does, it must contain the value of the token register.

Next, we make sure that the token register has contents.

```
        \expandafter\ifcsname footnotetoks\arabic{fncounter}\endcsname%
        \edef\tmp@@{\csname footnotetoks\arabic{fncounter}\endcsname}%
```

This `\detokenize` black magic tests if a token reg is actually empty[4]

```
        \expandafter\if\expandafter\relax\expandafter%
          \detokenize\expandafter{\the\tmp@@}\relax\else%
        \footnote@rule\gdef\footnote@rule{\ifvmode\else\par\fi}%
        \interlinepenalty\interfootnotelinepenalty%
```

Actually output the footnotes.

Having output the footnote, clear the token register (to save memory), then use `\let` to clear the definition of the macro. This ensures that we don't try to output the same footnote twice (at page end and entry end).

```
        \expandafter\the\tmp@@%
        \global\expandafter\tmp@@={}%
        \expandafter\let\tmp@@\@undefined%
      \fi\fi%
      \stepcounter{fncounter}%
      }%
  }%
  \endgroup%
}%
```

LaTeX works by building up a little more than a page, then calling the output routine. The output routine then decides where to put the page end from the built-up material. If there is a footnote mark, it could come after the page end, so we can't rely on the fact that there's footnote register to determine if we should output footnotes or not. The edge case is: if the footnote mark is held back for the next page, the footnote text would appear on the footer of the page where being built when the mark was expanded, which is the page before the footnote.

[4] `https://tex.stackexchange.com/questions/263733/whats-the-best-practice-way-to-test-whether-parameter-is-empty`

One fix for this is to have each footnote mark in the main body of the text output a `\mark` containing the footnote's counter value, and output the footnotes only to `\botmark`, which is the last `\mark` actually typeset on the page.

(see https://www.tug.org/TUGboat/Articles/tb11-4/tb30salomon.pdf, page 598, to read around footnotes).

`\outputfootnotes@endgentry` This macro is called at the end of each `gentry`. It switches into vertical mode (as ending `gentry` doesn't throw a `\break` or line end by itself), then we output all footnotes so far (by reading the footnote counter `\@mpfn`, as this is "sequential").

```
\newcommand{\outputfootnotes@endgentry}{%
  \if@gamebook@footnotes%
  \nopagebreak\ifhmode\\\fi% get into vertical mode
  \nopagebreak\outputfootnotes{\arabic{\@mpfn}}%
  \fi%
}%
```

`\noentryfoot` This method simply suppresses footnotes at the end of the entry within the current group. This forces footnotes within the current group to be printed at the bottom of the page.

This is useful in the case where LaTeX expands the footnote at the end of the entry, then decides to put the page split between the footnote mark and the text.

```
\newcommand{\noentryfoot}{\def\outputfootnotes@endgentry{}}
```

`\outputfootnotes@endpage` This is called by the output routine at the end of each page. If `\botmark` has a value, then we can output all footnotes up to that index.

```
\g@addto@macro{\gentry@footnotespergentry}{%
\newcommand{\outputfootnotes@endpage}{%
  \expandafter\if\expandafter\relax\expandafter%
  \detokenize\expandafter{\botmark}\relax\else%
  \outputfootnotes{\botmark}%
  \fi%
}}%
```

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.