

AcroTeX.Net

dps Package
Das Puzzle Spiel

D. P. Story

Abstract: Das Puzzle Spiel is a \LaTeX package for creating a puzzle, a message actually, and a series of questions and answers. The document consumer matches the questions with the answers. With each match, another letter appears in the puzzle. Upon completion of all questions, the message hidden in the puzzle is revealed. The game has an application as a classroom learning device.

Table of Contents

1 Introduction	4
2 Requirements	4
3 Comments on the demo files	5
4 Package Options	6
5 Designing your Puzzle	10
5.1 \DeclarePuzzle	10
5.2 Begin composing the questions and answers	12
6 Placing the Content	14
6.1 The title and instructions	15
6.2 The puzzle	15
6.3 The questions	15
6.4 The answers	16
6.5 The message field	17
6.6 Auxiliary files	17
7 Commands for controlling randomization	17
8 Printing and clearing the puzzle	18
9 Methods of handling long questions	18
9.1 The usebtnappr option	19
• The puzzle file	19
• The icons file	21
• The workflow to build the puzzle file	24
9.2 The uselayers option	24
• The puzzle file	24
9.3 Developing an end of game event	26
10 Creating a sideshow	26
10.1 Sideshow with the usebtnappr option	28
10.2 Sideshow with the uselayers option	29
11 Some small degree of security	29
12 Let's have some Fun	29
13 Checking for validity	30
14 Using the web Package	31
15 Thanks	31

Table of Contents (cont.)	3
---------------------------	---

16 Appendix	32
16.1 German Umlaut (dieresis)	32
16.2 Accents	32

1. Introduction

The work on this package was inspired by one of my son's worksheets in pre-algebra. The worksheet consisted of a series of simple algebraic expressions the student was to simplify. The simplified form was listed somewhere in the answers column. The answer had a letter associated with it which the student then placed in a puzzle. Upon completion of the worksheet, the puzzle (message) is completely filled in; if the message makes sense, the student can determine that he/she did the worksheet correctly.

I set out to duplicate this worksheet for electronic media, but also to have an option for paper as well.

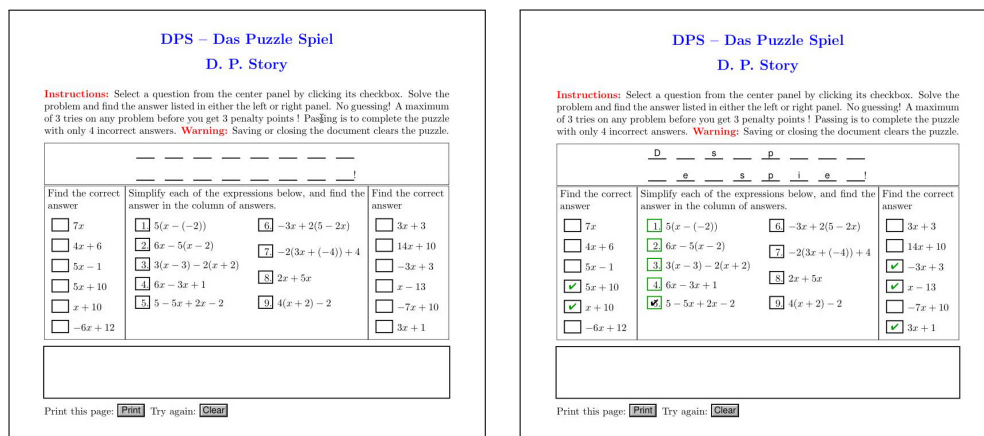


Figure 1: dps_demo.pdf: Initial layout (left) and partially worked (right)

2. Requirements

The following packages are required for dps beyond that of the standard \LaTeX distribution:

- **AeB:**¹ The web and eforms components of the Acro \TeX eEducation Bundle are used.
- **random.tex:**² A \TeX / \LaTeX macro file to generate random numbers, the package was written by Donald Arseneau.
- If the usebtapp option is taken, the icon-appr package, dated 2020/06/05 or later, is required.³

The package should work for users of dvips \rightarrow distiller, pdflatex, lualatex, and xelatex. A document author who owns the Acrobat application and plans to use either the usebtapp or uselayers option should have the aeb_pro package (completely) installed.

¹<https://ctan.org/pkg/acrotex>

²<https://ctan.org/pkg/random>

³<https://ctan.org/pkg/icon-appr>

3. Comments on the demo files

The distribution comes with the following demonstration files.

- basic folder*
- `examples/basic/dps_demo.tex`: A demo file that you can use to try the package with different options. Build the source file for screen or for paper. Two column format for questions, and half the answers on the left, and half on the right.
This demo file has more answers than questions.
 - `examples/basic/dps_d1.tex`
The original file constructing during the development of this package. This puzzle has the famous u-umlaut. Can be compiled with the `forpaper/forcolorpaper` option of `web`. The layout is designated as “design 1” (d1): questions in center in two column format and answer on the left and right.
This demo file illustrates the special name of `cr`, as well as `space` and `punc`.
 - `examples/basic/dps_d1_p.tex`
Same as `dps_d1.tex`, but set up as a paper document (using the `forcolorpaper` option). The font size is enlarged to make it easier for filling the puzzle out using a pencil.
 - `examples/basic/dps_d2.tex`
Same puzzle/questions as `dps_d1.tex`, but uses the layout designated as “design 2” (d2): Questions and answer in column format on left and right, puzzle in the center. This leaves a lot of white space in the middle, perhaps for a graphic. This example also manually inserts the answer key command `\AnswerKey`, which manifests itself when the `showletters` option is taken.
 - `examples/basic/dps_d3.tex`
Same puzzle/questions as `dps_d1.tex`, but uses the layout designated as “design 3” (d3): Puzzle and questions vertically aligned (questions in two column format), answer in single column. This file uses many of the formatting commands mentioned in the documentation.
 - `examples/basic/dps_signin.tex`
Demonstrates how to require the player to enter his/her name, useful when puzzle is to be handed in for extra credit.
 - `examples/basic/dps_demo.tex`
Indicates how to change the appearance of some of the form elements of the puzzle.
 - `examples/basic/stat_match1.tex`
An example of a puzzle with extended length questions. Demo can be used as a paper assignment or a digital assignment.
 - `examples/basic/stat_match1-print.tex`
A layout with long questions designed for paper.

There are several “advanced examples” that demonstrate two methods of posing *extended length questions*; these two methods correspond to the two options `usebtnappr` and `uselayers`.

- usebtnappr* folder
- `examples/advanced/usebtnappr/basic/stat_match1.tex`
Uses the `usebtnappr` to create icon pushbutton appearances of the extended length questions. This example works for all common workflows: `pdflatex`, `lualatex`, `xelatex`, and `dvips -> distiller`.
 - `examples/advanced/usebtnappr/sideshow/first_date.tex`
A file developed from by cyber buddy, who I've never met, to offer advice for going on a date *mit eine Fräulein*. This example works for all common workflows: `pdflatex`, `lualatex`, `xelatex`, and `dvips -> distiller`. Uses a graphical `sideshow`.
- uselayers* folder
- `examples/advanced/uselayers/basic/stat_math1-tb.tex`
Uses the `uselayers` option and the `textpos` package.
An extra credit assignment given to my statistics class in 2006. The questions are placed in separate layers and appear when a question checkbox is selected; this allows for more wordy questions without taking up a lot of space in the question part of the puzzle.
 - `examples/advanced/uselayers/basic/stat_math1-ep.tex`
Same as puzzle as `stat_math1-tb.tex` but uses the `eso-pic` package.
 - `examples/advanced/uselayers/basic/stat_math1_g.tex`
Same as a `stat_math1-tb.tex`, but more graphical. If memory serves, this version was developed my good cyber friend Jürgen.
 - `examples/advanced/uselayers/sideshow/first_data.tex`
A file developed from by cyber buddy, who I've never met, to offer advice for going on a date *mit eine Fräulein*. Uses a graphical `sideshow`.
 - `examples/advanced/uselayers/sideshow/first_data_g.tex`
More graphical version of `first_data.tex`, but with different design and graphical `sideshow`.

4. Package Options

Here we list the options of package `dps`.

`nonrandomized`: The default behavior is to randomize the questions and to randomize the answers. With this option, the questions and answer are listed in the order that they appear in the `Composing` environment. This makes it easy for the document author to quickly solve the puzzle and to see if the check marks and letters appear as they should.

`!nonrandomized`: (Convenience option) Cancels the effects of `nonrandomized`; in this case, the answers are randomized (the default).

`viewmode`: Useful when using a dvi previewer or a PDF previewer. Here you can see the placement of the puzzle (with the letters to the puzzle filled in) and the boxes where the checkboxes go. Useful in the designing the layout of your game phase.

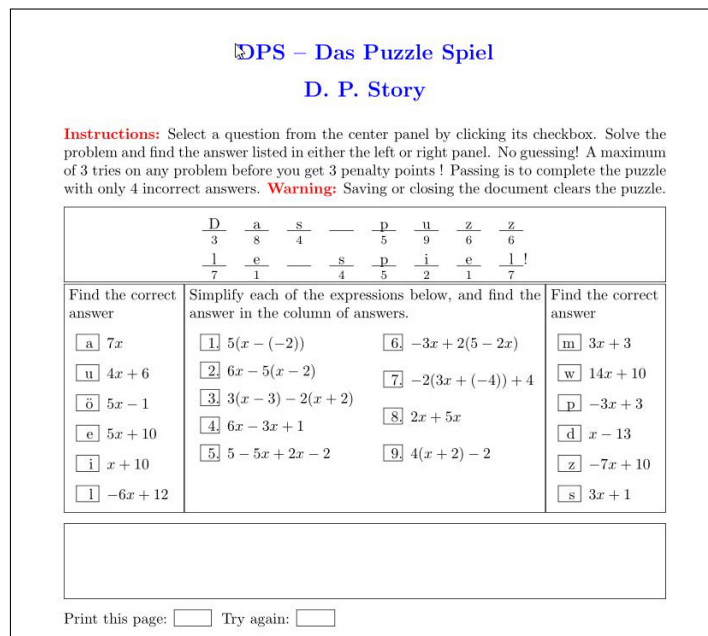


Figure 2: Compiled with `viewmode` option

`!viewmode`: (Convenience option) Cancels the effects of `viewmode`, this same as not specifying `viewmode`.

`showletters`: Sets up a visual relation between the questions, the answers and the puzzle elements. Useful in design phase, can be used with `viewmode`. Refer to [Figure 3](#).

`\AnswerKey`

When `showletters`, the command `\AnswerKey` is populated with the answer key to the puzzle. `\AnswerKey` is automatically inserted into the running footer when the `showanswerkey` option is taken; otherwise, it can be manually inserted into the document by expanding `\AnswerKey` in a location somewhere *after* the building of the puzzle, the questions and the answers.

`!showletters`: (Convenience option) Cancels the effects of `showletters`; in this case, the letters are not shown (the default).

`showanswerkey`: Shows the answer key in the footer of the document. If the `graphicx` package is loaded, the answer key is rotated 180°. This option is meant to be used when the `forpaper` option is taken in the `web` package. Refer to [Figure 4](#).

`\setdpsfootskip`

The positioning of the running footer may be adjust using the convenience command `\setdpsfootskip{<skip>}`. When there is no paper option for web (neither

DPS – Das Puzzle Spiel

D. P. Story

Instructions: Select a question from the center panel by clicking its checkbox. Solve the problem and find the answer listed in either the left or right panel. No guessing! A maximum of 3 tries on any problem before you get 3 penalty points! Passing is to complete the puzzle with only 4 incorrect answers. **Warning:** Saving or closing the document clears the puzzle.

	$\begin{array}{cccccc} \frac{3}{7} & \frac{8}{1} & \frac{4}{4} & \frac{5}{5} & \frac{9}{2} & \frac{6}{1} & \frac{6}{7} \end{array}$	
Find the correct answer	Simplify each of the expressions below, and find the answer in the column of answers.	Find the correct answer
<input type="checkbox"/> a $7x$ <input type="checkbox"/> n $4x + 6$ <input type="checkbox"/> ö $5x - 1$ <input type="checkbox"/> e $5x + 10$ <input type="checkbox"/> i $x + 10$ <input type="checkbox"/> l $-6x + 12$	<input type="checkbox"/> 1 $5(x - (-2))$ <input type="checkbox"/> 2 $6x - 5(x - 2)$ <input type="checkbox"/> 3 $3(x - 3) - 2(x + 2)$ <input type="checkbox"/> 4 $6x - 3x + 1$ <input type="checkbox"/> 5 $5 - 5x + 2x - 2$	<input type="checkbox"/> 6 $-3x + 2(5 - 2x)$ <input type="checkbox"/> 7 $-2(3x + (-4)) + 4$ <input type="checkbox"/> 8 $2x + 5x$ <input type="checkbox"/> 9 $4(x + 2) - 2$ <input type="checkbox"/> m $3x + 3$ <input type="checkbox"/> w $14x + 10$ <input type="checkbox"/> p $-3x + 3$ <input type="checkbox"/> d $x - 13$ <input type="checkbox"/> z $-7x + 10$ <input type="checkbox"/> s $3x + 1$
Print this page: <input type="button" value="Print"/> Try again: <input type="button" value="Clear"/>		

Figure 3: Compiled with showletters option

forpaper or forcolorpaper), `<skip>` is the distance up from the bottom edge of the screen page. The default is `\setdpsfootskip{.25in}`. When there is a paper option, this command does nothing; location of the running footer is determined by the \LaTeX `\footskip` register.

`!showanswerkey:` (Convenience option) Turns off the showanswerkey option.

`savedata:` Saves three pieces of information to the local hard drive : (1) the seed value used by the random package to randomize the questions and answers; (2) the value of the last number generated; and (3) the answer key. This information is saved to the file `\jobname_data.sav`.

- The seed value can be used to reproduce the exact randomization at a later time. Open the file `\jobname_data.sav` in your editor, it might look like this:

```
\randomi=482053344 % Initial seed:
\dpsLastSeed{271256117}
% Answer Key: 1--e; 2--s; 3--i; 4--u; 5--l; 6--d; 7--a; 8--z; 9--p;
Copy the number indicated as the Initial seed and paste it into the argument
\useRandomSeed{482053344}. Place this command in the preamble, below
the \usepackage{dps} line. This should reproduce the same randomiza-
tion.
```


DPS – Das Puzzle Spiel

D. P. Story

Instructions: Select a question from the center panel by clicking its checkbox. Solve the problem and find the answer listed in either the left or right panel. No guessing! A maximum of 3 tries on any problem before you get 3 penalty points! Passing is to complete the puzzle with only 4 incorrect answers. **Warning:** Saving or closing the document clears the puzzle.

	$\frac{3}{7} \quad \frac{8}{1} \quad \frac{4}{4} \quad \frac{5}{5} \quad \frac{9}{2} \quad \frac{6}{1} \quad \frac{6}{7}$	
Find the correct answer	Simplify each of the expressions below, and find the answer in the column of answers.	Find the correct answer
<input type="checkbox"/> a $7x$ <input type="checkbox"/> n $4x + 6$ <input type="checkbox"/> o $5x - 1$ <input type="checkbox"/> e $5x + 10$ <input type="checkbox"/> i $x + 10$ <input type="checkbox"/> l $-6x + 12$	<input type="checkbox"/> 1 $5(x - (-2))$ <input type="checkbox"/> 2 $6x - 5(x - 2)$ <input type="checkbox"/> 3 $3(x - 3) - 2(x + 2)$ <input type="checkbox"/> 4 $6x - 3x + 1$ <input type="checkbox"/> 5 $5 - 5x + 2x - 2$	<input type="checkbox"/> 6 $-3x + 2(5 - 2x)$ <input type="checkbox"/> 7 $-2(3x + (-4)) + 4$ <input type="checkbox"/> 8 $2x + 5x$ <input type="checkbox"/> 9 $4(x + 2) - 2$
		<input type="checkbox"/> m $3x + 3$ <input type="checkbox"/> w $14x + 10$ <input type="checkbox"/> p $-3x + 3$ <input type="checkbox"/> d $x - 13$ <input type="checkbox"/> z $-7x + 10$ <input type="checkbox"/> s $3x + 1$

Print this page: Try again:

!n-6 !e-8 !t-7 !z-9 !d-g !s-t !p-e !t-z !s-t

Figure 4: Compiled with showanswerkey option

- The second line is use when the `\useLastSeed` command appears in the preamble. This is the last number generated by the current compile.
- The third line in the above verbatim listing is the answer key, for the randomization initiated by the seed on the first line. You can copy and paste this second line into the \TeX/\LaTeX document and publish the solution in a separate document, at a later time. This is useful when publishing for paper.

`!savedata:` (Convenience option) Turns off the savedata option.

`usebtnappr` Use this option to provide support for longer questions without taking up anymore space on the digital page. The option uses icon appearances of a push button. Refer to Section 9.1.

`uselayers` Use this option to provide support for longer questions without taking up anymore space on the digital page. The option places the long question in separate layers (OCGs). Refer to Section 9.2.

`lang=english|german|custom:` Currently, there are only two language options. The value of `custom` allows you to create your own language strings.

To use the `lang=custom` option, you must create the file `dps_str_cus.def`. Do this by taking the file `dps_str_us.def` (or `dps_str_de.def`, if you prefer), copying the file and changing its name to `dps_str_cus.def`. Open `dps_str_cus.def`

in your favorite editor and change the text strings to a language of your choice, or change the strings that tickle your funny bone more than the ones provided.⁴ To represent accented characters, use the octal encoding defined in `pd1enc.def`, as distributed with the `hyperref` bundle. The u-umlaut, for example, should appear in the file as `\string\374` and not as `\"u`.

Change notification: If you have already written your own `dps_str_cus.def` for an earlier version of `dps`; there is a change you should attend to: In the definition of `\regretPleased` change `(nMissed>n)` to read `(nMissed>nPassing)`.

- For your convenience, the ‘Appendix’ on page 32 contains a listing of the octal codes for accented letters. Here is an important option of `web` package.

`forpaper/forcolorpaper`: These are options of the `web` package, not the `dps` package. Then this option is taken, the puzzle created is meant for paper publication. No Acrobat forms will be created, the `showletters` option of `dps` is automatically taken. If you originally designed the game for the screen, you may have to rework the sizing and design of the game to fit everything into the constraint of a piece of paper. Landscape is an option to think about, depending on your design and the number of questions and answers.

5. Designing your Puzzle

There are several test files that you can use as basis of constructing your own puzzle.⁵ The files themselves illustrate adequately the structure of the puzzle, but a few remarks are in order.

5.1. `\DeclarePuzzle`

The first step is to have a message, either funny, serious, or whatever. For the purpose of illustration, suppose your message is “Hello, Jürgen!” This message has all the elements that need to be discussed, letters, punctuation, spaces, capitalized letters, and accented letters.

`\DeclarePuzzle{<puzzle-args>}`: The `<puzzle-args>` consists of a series of *paired parameters*:

```
\nPuzzleCols{<nCols>} % optional
\DeclarePuzzle{%
  {<letter1>}{<name1>}
  {<letter2>}{<name2>}
  ...
  ...
  {<lettern>}{<namen>}
}
```

⁴Or tickle your funny bone *less*, if you are a crusty one.

⁵My apologies, this game is more properly described as a matching game with message, but my friend insisted that I call it a puzzle so that the package could be named Das Puzzle Spiel, or `dps`, which are my initials. ☹

`\nPuzzleCols{<nCols>}` is a convenience command for setting the number of columns in the puzzle; `<nCols>` is passed to `\insertPuzzle{<nCols>}`, which appears in the body of the document.

Parameter description for `\DeclarePuzzle`. The argument `<letter>` represents a letter in the puzzle; `<letter>` plays two roles: (1) it is used to typeset the letters into the document when certain options, such as `viewmode`, are used; (2) it is used as the default value of a text fields that is created (when the puzzle is built to be interactive). This creates a problem for special characters, such as `ü`; it is a typeset letter, and is `\string\374` when placed into a text field (`\341`) is the (octal) PDFDocEncoding of u-umlaut. The way around this conundrum is to use `\texorpdfstring`: use `<letter>` to be,

```
\texorpdfstring{\protect\"{u}}{\ifxetex ü\else\string\374\fi}
```

Notice the use of `\protect` to protect this moving argument. Also included above is a special case for `xelatex`, which does not support the octal notation; in this case, using the capabilities of your editor, simply type in an u-umlaut. The above complex expression is normally not needed. Here, we try to provide guidance for all PDF creators. If you stick to one, the `\ifxetex` conditional is not needed.

The second argument pair is `<name>`, this is a unique name that is used in the construction of the underlying text field name: the field name becomes `puzzle.<name>`. As a result, `<name>` needs to be a JavaScript identifier (or, basically consist of letters and numbers). In the case of special characters such as our umlaut problem, we can assign a name like so:

```
\texorpdfstring{\protect\"{u}}{\ifxetex ü\else\string\374\fi}{uml}
or
\tops{\protect\"{u}}{\ifxetex ü\else\string\374\fi}{uml}
```

`\tops` where `\tops` is an alias for `\texorpdfstring`. This argument pair is seen several times in the demonstration files. There are three special names, these are `space`, `punc`, and `cr`; as a argument pair, these should appear as follows: `{ }{space}`, `{ , }{punc}`, and `{ }{cr}`, respectively. Spaces and punctuation are not normally part of the puzzle to be discovered by answering questions, though they could be. The special name `cr` terminates a row.

In the design of the puzzle there are three sets of form fields to manage: checkboxes for the questions, checkboxes for the answers, and text fields for the puzzle. The checkboxes and puzzle letter are all tied together by a common base field name, which is the second parameter in the parameter pairs.

In our puzzle, “Hello, Jürgen!”, we place the `\DeclarePuzzle` in the preamble, as shown in [Figure 5](#). For letters, the second parameter in the pair can simply be the same, as in `{H}{H}` and `{e}{e}`.

Rule: As a general rule, and this rule will be repeated later, there should be one question for each distinct second argument (`<name>`). There should be *no question* corresponding to the names `space`, `punc`, and `cr`.

```

\DeclarePuzzle{%
  {H}{H}
  {e}{e}
  {l}{l}
  {l}{l}
  {o}{o}
  {,}{punc}
  {}{space}
  {J}{J}
  {\tops{\protect\"{u}}%
   {\ifxetex ü\else\string\374}\fi}{uml}
  {r}{r}
  {g}{g}
  {e}{e}
  {n}{n}
  {!}{punc}
}

```

Figure 5: The “Hello, Jürgen” Puzzle

In the example of Figure 5, letters like e and l appear more than once in the puzzle. Note that the second argument of each of these two e’s is the same. There should be only one for this e, and when the question associated with e is correctly answered, both e’s in the puzzle will appear. (If you want a question for each e, then you need to name the fields differently, {e}{e1} and {e}{e2}, for example.)

5.2. Begin composing the questions and answers

We create questions and answers for the puzzle within the `Composing` environment. Questions are posed within the `cQ` environment and answers are written within the `cA` environment.

```

\DeclarePuzzle{<puzzle-args>}
\begin{Composing}
\begin{cQ}{<name>}
  <some question>
\end{cQ}
\begin{cA}[<alt-letter>]{<name>}
  <some answer>
\end{cA}
...
\end{Composing}

```

Composing: You compose the questions and answers within the `Composing` environment. At the top of the environment, certain counters are initialized. All the work is done at the end of the environment: The number of questions and answers are known, at which point, the order of the questions and answers are randomized.

The `Composing` environment follows `\DeclarePuzzle`.

`cQ` and `cA`: Within the `Composing` environment, you compose your questions and answers within the `cQ` and `cA` environments, respectively. Each question must be followed by its answer. You can have more answers than questions, but these answers *must* be listed last.

Each of these environments has one required argument, and `cA` has one optional argument. The required argument, $\langle name \rangle$, is the field name to which this answer corresponds. (The second argument of the paired arguments of `\DeclarePuzzle`.)

This optional argument, $\langle alt-letter \rangle$, of the `cA` environment is only relevant when the document is compiled with the `showletters` option. The value of the argument is a letter to appear in the answers column (and in the answer key). Normally, one of the first entries in the `\DeclarePuzzle` command is used. Cases where you would want to include this optional argument are,

- (1) when giving an answer that does not correspond to a question. For example, in `dps_demo.tex`, there are several answers that are distractions

```
\begin{cA}[w]{fake1}
$14x+10$
\end{cA}
```

In the puzzle this answer appears in the list of answers but does not correspond to any question. The letter associated with this answer is ‘w’.

- (2) the letter is capitalized, suggesting a proper name or the beginning of a sentence, use the optional argument to list the letter in lower case. For example,

```
\begin{cA}[h]{H}
$-2x-2$
\end{cA}
```

Easily set up the `Composing` environment. Once you decide on your puzzle (this is easy), you need to set up the corresponding environments `Composing`, `cQ`, and `cA`.

Rule: As a general rule, and this rule will be repeated later, there should be one question for each distinct second argument, that is, for each distinct $\langle name \rangle$, with the exception of the names `space`, `punc`, and `cr`.

Because of human errors, sometimes we have questions/answer pairs that correspond to a duplicate field name—perhaps this letter occurs multiple times. No, no, that violates the red rule above. After having made this same error several times myself, I decided to let \TeX do the work for me.

Just after the end of the argument of `\DeclarePuzzle`, prior to authoring the questions and answers, place the `\writeComposingEnv`:

```
\writeComposingEnv
```

Remember, `\DeclarePuzzle` is in the preamble and the beginning of document has not been encountered; the command writes an outline for the puzzle—based on the

arguments of `\DeclarePuzzle`—to the file `\jobname_comp.def` and ends the document. Once this file is created, copy and paste its contents into your source file. It should look like this:

```
\begin{Composing}

\begin{cQ}{M}
\end{cQ}
\begin{cA}{M}
\end{cA}

\begin{cQ}{a}
\end{cQ}
\begin{cA}{a}
\end{cA}
...
\end{Composing}
```

The correct number of environments should be there, with the correct argument inserted for each environment. Cool! Now, just compose your questions and answers.

After `\jobname_comp.def` is created and copied into the puzzle document, *delete* the command `\writeComposingEnv`. This command is not part of the puzzle, but a helper command. As an option, rather than pasting the contents into your document following the `\DeclarePuzzle`, you can fill in your questions and answers to your puzzle within the `\jobname_comp.def`, input the file `\jobname_comp.def` with `\input{\jobname_comp.def}`. **Warning:** If you leave your questions and answers in the auxiliary file `\jobname_comp.def` you may overwrite them by un-commenting `\writeComposingEnv` and compiling your puzzle document.

Another rule:

Rule: You must have one correct answer per question. Each answer is unique in the list of all answers (no two answers can be the same). You can have more answers than questions (distractions—answers “close” in appearance to the correct answer). Each of the distractions must have a unique *<name>*.

6. Placing the Content

The content consists of five parts, plus whatever you wish to include in the document:

1. The title and instructions.
2. The puzzle
3. The questions
4. The answers
5. The message field

Each of these is discussed in the sections that follow.

6.1. The title and instructions

The title and instructions are your bailiwick, see package demo files for suggestions.

6.2. The puzzle

`\insertPuzzle{empty|<nCols>}`: The puzzle, which consists of the first of the paired arguments declared in the `\DeclarePuzzle` command, is laid out in a tabular format. The one required argument of `\insertPuzzle` is either the empty argument `{}` or `<nCols>`, the number of columns per row to be used.

If you say `\insertPuzzle{}`, then `\nPuzzleCols{<nCols>}` is expected to appear in the preamble. If the argument of `\insertPuzzle` is empty and `\nPuzzleCols` does not appear in the preamble, a warning is issued and `<nCols>` is set to 10.

In the demo files, `\insertPuzzle` is enclosed in a `minipage` environment (and in a `\fbox` as well). By placing `\insertPuzzle` in this way, you can control the width of the table, and it may help fit it with the other components of the game.

`\PuzzleAppearance`: Use this command to change the appearance of the Acrobat text field that comprise the interactive puzzle. The command takes one argument, this argument consists of one or more commands from the eForms package that change the appearance of a field. For example,

```
\PuzzleAppearance{\BC{1 0 0}}
\PuzzleAppearance{\BC{red}} % if xcolor is loaded
```

changes the boundary color to red. See the eforms documentation.

`\rowsep{<skip>}`: By setting `\rowsep`, you can adjust the vertical space between tabular rows. The command takes one argument, the amount of vertical skip, for example,

```
\rowsep{2ex}
```

`\wdPuzzleFields{<length>}` Sets the width of a puzzle field to `<length>`. The default is 1.6em

`\htPuzzleFields{<length>}` Sets the height of the puzzle field to `<length>`. The default is 11bp.

6.3. The questions

`\displayRandomizedQuestions`: The questions are inserted by this command. This command must be placed in an `enumerate` environment. This will number the questions, so when, for example, the `showletters`, discussed later, is taken, there is a visual mapping between the questions, the answers and the puzzle.

If the number of questions is not great, you can list the questions in a single column; however, in the examples that I have done, I've determined that a two column format (using the `multicol` package) seems to me to be the best layout for the questions.

`\QuesAppearance`: Use this command to change the appearance of the Acrobat checkboxes for the questions that appear in the label margin. The command takes one argument, this argument consists of one or more commands from the eForms package that change the appearance of a field. For example,

```
\QuesAppearance{\BC{gray}} % assumes xcolor pkg or just \BC{.5 .5 .5}
```

changes the boundary color to a gray color. See the eforms documentation.

`\widestFmtdQNum{<str>}` Sets the width of the checkbox for the questions. The argument `<str>` is a string whose width determines the width of the checkboxes. The default is `\widestFmtdQNum{00.}`. If the numbers are typeset in bold, then `\widestFmtdQNum{\textbf{00.}}` is a little wider to account for the bold font.

`\htOfQ{<length>}` Sets the height of the checkbox for the question. The default is `\htOfQ{13bp}`.

6.4. The answers

`\displayRandomizedAnswers`: As with the questions, the answer are displayed in by a similar command. Again, this command should be in a list environment, preferably the `itemize`. The list label is suppressed by placing `\item[]` before each answer. When the `showletters` option is taken, the letter this answer corresponds to will be the label.

One of the design layouts in the demo files has the questions in a two column format in the center with two columns of answers, half the answers to the left of the questions, and the other half to the right. The two commands

```
\displayRandomizedAnswersLeftPanel
\displayRandomizedAnswersRightPanel
```

are used for that purpose.⁶ As with `\displayRandomizedAnswers`, each of these commands should be in an `itemize` environment.

`\AnsAppearance`: Use this command to change the appearance of the Acrobat checkboxes for the answers that appear in the label margin. The command takes one argument, this argument consists of one or more commands from the eForms package that change the appearance of a field. For example,

```
\AnsAppearance{\BC{gray}} % assumes xcolor pkg, or \BC{.5 .5 .5}
```

changes the boundary color to a gray color. See the eforms documentation.

`\ltrFmtA{<\cmd{#1}>}` Formats the letters in the list of answers when the option `showletters` is active. In the argument, `#1` references the current letter to be typeset. For example, `\ltrFmtA{\textbf{\textcolor{blue}{#1}}}` typesets the letters in bold and blue.

⁶TeX doesn't know his left from his right, so you can actually place the left panel listing on the right. TeX will not object!

`\widestFmtdALtr{<str>}` Sets the width of the checkbox for the answers. The argument `<str>` is a string whose width determines the width of the checkboxes. The default is `\widestFmtdALtr{w}`.

`\htOfA{<length>}` Sets the height of the checkbox for the answer. The default is `\htOfA{13bp}`.

6.5. The message field

`\placeMessageField[<opts>]{<wd>}{<ht>}`: This Acrobat text field is used to write messages to the user. If the user tries to choose an answer before selecting an answer, s/he gets the message

"You must choose a question to answer before you answer!"

For the interactive version of this game⁷, there is a language option for `dps` to change the messages from English, the default, to German, for example.

The parameters are `<wd>` (the width of the text field), `<ht>` (the height of the text field and `<opt>` (the optional argument for changing the appearance of the field, as described in the documentation of the `eforms` package.

The message field is automatically removed when the document is compiled in the `forpaper` option of the web package.

6.6. Auxiliary files

Developing a puzzle file creates a number of auxiliary files, in addition to the usual ones of a typical \LaTeX file. The `dps` package creates a large number of CUT files and a few SAV files. If you create a sideshow (page 26), additional PDF files are created as well. As a general rule, do not delete these files until you are finished building your puzzle and you have verified that is working correctly.

7. Commands for controlling randomization

When randomizing is to be used (that is, the option `!nonrandomized` is in effect, or the `nonrandomized` option is not specified), there are several commands to control the initial seed of the random number generator.

```
\useRandomSeed{<seed>} % eg, \useRandomSeed{187968637}
\inputRandomSeed
\useLastSeed
```

All commands are placed in the preamble. Only one of the three should appear during any compile.

`\useRandomSeed{<seed>}` initializes the random number generator to `<seed>`.

⁷Does that mean there is also a non-interactive version of this game, sir? Yes, yes there is. ~~DS~~

`\inputRandomSeed` inputs the seed value saved by the `savedata` option; as a result, the same randomization sequence is obtained each time the file is compiled.⁸ In effect, this command freezes the randomization sequence as long as the file `\jobname_data.sav` has not been deleted. To continue to use this random sequence after `\jobname_data.sav` is deleted, save the value of the initial seed first, as described in the description of the `savedata` option on [page 8](#).

`\useLastSeed` initializes the random number generator with the last random number generated in the *previous* compile. In this way, you get a new randomization each time you compile the file.

The last two commands assume `\jobname_data.sav` exists; otherwise they take an initial seed based on the current date and time.

8. Printing and clearing the puzzle

The `dps` package provides the following two commands, which produce push buttons:

```
\printDPS[⟨form-options⟩]{⟨wd⟩}{⟨ht⟩} % prints the document
\resetDPS[⟨form-options⟩]{⟨wd⟩}{⟨ht⟩} % clears the document
```

The `⟨wd⟩` and `⟨ht⟩` are the width and height of the push button; the optional argument `⟨form-options⟩` are key-value pairs to change the appearance of the buttons. Familiarity with the `eforms` package is needed.

The `dps` provides the command `\dpsResetHook` to add JavaScript lines to the action of `\resetDPS`. The command takes one argument to pass the JavaScript lines to `\resetDPS`; for example, `\dpsResetHook{\dpsHideFld("btnEmoji");}` appears in several of the example file. This hides the `btnEmoji` image.

9. Methods of handling long questions

All the basic examples (those in the folder `examples/basic`) feature *short questions* that conveniently fit into the space provided.⁹ More complex questions require longer questions. The problem, then, is to develop a method of posing long questions, without disturbing the puzzle design, for an *interactive puzzle*.

In this section, we detail two methods of posing long questions, while saving space on a one page puzzle document; these are (1) place the long questions into an appearance of an push button (yes, you can do that); and (2) place the long questions into a layer (optional content group, OCG). All \LaTeX workflows are supported by method (1), while method (2) requires the `dvips -> distiller` workflow.

interactive puzzles Both methods are designed for an interactive puzzle, not a paper puzzle.

⁸This assumes that randomized items are neither added or deleted.

⁹With the one exception of `stat_match1-print.tex` in the `basic` folder.

9.1. The `usebtnappr` option

Demo file. The demonstration file for this subsection is `stat_match1.tex`, found in the `examples/advanced/icon-appr` folder.

This solution requires two files: (1) the puzzle file; and the (2) icon file.

- **The puzzle file**

Within the puzzle file, the `usebtnappr` option is specified. The `\usepackage` command for `dps` below specifies the minimal options:

```
\usepackage[%
  usebtnappr,
  nonrandomized,
  savedata
]{dps}
```

In the preamble. We place two environments; `embedding`, to embed the icons; and `setContent`, to pose the long questions.

icon-appr package

Embedding the icons. The `usebtnappr` brings in the `icon-appr` package,¹⁰ which defines the `embedding` environment and the `\embedIcon` command. The `dps` package defines `\dpsEmbedIcons`. This command embeds all the dynamically created PDF graphics that comprise the long questions. `\dpsEmbedIcons` is followed by other optional `\embedIcon` commands.

```
\begin{embedding}
\dpsEmbedIcons
<other-embeds>
\end{embedding}
```

Below is the example from the demo file.

```
\begin{embedding}
\dpsEmbedIcons
\embedIcon[name=Emoji,placement=btnEmoji]{MyEmoji.pdf}
\end{embedding}
```

The second line is the optional one; here, we explicitly embed, using the syntax of `icon-appr`, an additional graphic for use by the puzzle.

Special note for xelatex users. The `embedding` environment is placed in the preamble but the indirect references (a PDF term) of the embedded graphics are not calculated until the first page is shipped out (a \LaTeX term); therefore, the puzzle *should be on the second page* when the `textpos` package is used to position the graphics. This issue does not arise when using the `eso-pic` package.

textpos pkg

¹⁰<https://ctan.org/pkg/icon-appr>

Posing long questions. Following the `embedding` environment comes the usual content for designing the puzzle: the `\DeclarePuzzle` data structure, followed by the `Composing` environment. It is within the `cQ` environment of the `Composing` environment that there is a change in content.

posing long question The `usebtnappr` option defines the `setContent` environment, which is placed in the `cQ` environment.

```
\begin{cQ}{\langle name \rangle}
  \langle question-prompt \rangle
\begin{setContent}{\langle name \rangle}
  \langle long-question \rangle
\end{setContent}
\end{cQ}
```

The `\langle name \rangle` argument of `setContent` is same as the argument of the enclosing `cQ` environment. `setContent` is a verbatim write environment; `\begin{setContent}` can follow the end of the `question-prompt` (one or more words that suggest what the question is about), but the `\end{setContent}` must be in the left-most margin, as shown above. We illustrate first with an example taken from the demo file:

```
\begin{Composing}
\begin{cQ}{R}
Branches of Statistics\begin{setContent}{R}
The two branches of statistics are descriptive and
\underbar{\hspace{.5in}}.
\end{setContent}
\end{cQ}
\begin{cA}[r]{R}
inferential
\end{cA}
...
\end{Composing}
```

A long question is not required, in the demo file most have long questions and some do not.

The `setContent` environment writes its content verbatim to a CUT file (named `\jobname-sc(\langle num \rangle).cut`). For example, the CUT file for the first question of the demo file reads,

```
\textbf{Problem 1}\newline
The two branches of statistics
are descriptive and \underbar{\hspace{.5in}}.
```

The first line seen above can be modified using the following two commands.

```
\newcommand{\quesNumTxt}[1]{\protect\textbf{Problem #1}}
\newcommand{\quesNumTxTPost}{\protect\newline}
```

These may be redefined.

On the puzzle page. Aside from the layout of the puzzle, questions, and answers commands, above all these, place the following commands:

```
\placeQuesIcon{\placement-cmds}
\placeOtherIcon{\placement-cmds}
```

*eso-pic, textpos
packages*

The nature of the $\langle placement-cmds \rangle$ depends on the “placement” package used. The demo files use either the *eso-pic* or *textpos* package. The $\langle placement-cmds \rangle$ place the text field $\dpsQuesIcon\{#1\}\langle wd \rangle\langle ht \rangle$. The following is taken from the demo file, it uses the *eso-pic* package.

```
\placeQuesIcon{\AddToShipoutPictureFG*\AtTextCenter{\put(-72,0)
{\dpsQuesIcon\{#1\}\{2.25in\}\{9\baselineskip\}}}}
```

If other icons were embedded in the `embedding` environment, those icons can be placed using the `\placeOtherIcon` command.

```
\placeOtherIcon{\AddToShipoutPictureFG*\AtTextCenter{\put(-72,0)
{\dpsOtherIcon[\I{\csOf{Emoji}}]{btnEmoji}\{2.25in\}\{9\baselineskip\}}}}
```

• The icons file

The `usebtnappr` option requires the use of a second file, named `icons.tex`.¹¹ The `icons.tex` file in the `examples/advanced/icon-appr`. The icons file (`icons.tex`) is a very short file and is placed in the same folder as the puzzle file.

```
\documentclass{article}
\usepackage[!useacrobat]{icon-doc}
\margins{3pt}{3pt}{3pt}{3pt} % left,right,top,bottom (web command)
\screensize{9\baselineskip}{2.25in} % height,width (web command)
\begin{document}
\small
\createRequiredIcons{12}{stat_match1}
\end{document}
```

icon-doc package

The icons file is placed in the same folder as the puzzle file. The `dps` distribution has the `icon-doc` package, a short package designed for the icons file. The package has two options `useacrobat` and `!useacrobat`, the default is `!useacrobat`. Document authors who use `pdflatex`, `lualatex`, or `dvips -> distiller` need not bother with this option; the option is designed for `xelatex` users.

Use the `\margins` command to adjust the boundary margins of icons file. Use the `\screensize` to adjust the height of the icons file and the width of the icons file. The dimensions shown above are the ones used by the demo file.

The `\createRequiredIcons\{<num-ques>\}\{<puzzle-file>\}` command has two arguments: $\langle num-ques \rangle$ is the number of questions in the puzzle file; $\langle puzzle-file \rangle$ is the base name of the puzzle file. The command has two behaviors:

1. **For xelatex authors that do not have Acrobat.** When the icon file is compiled, the result is $\langle num-ques \rangle$ PDFs, each PDF contains a *single* long question. Since the author does not have Acrobat, the icons file is compiled with the `!useacrobat`

shellesc pkg

option. This case requires the *shellesc* package and that your \LaTeX editor be setup to use the `shell-escape` switch of your editor. Refer to Figure 6 to see how to do this for the WinEdt editor. Other editors/ \TeX systems may support the `shell-escape` switch.

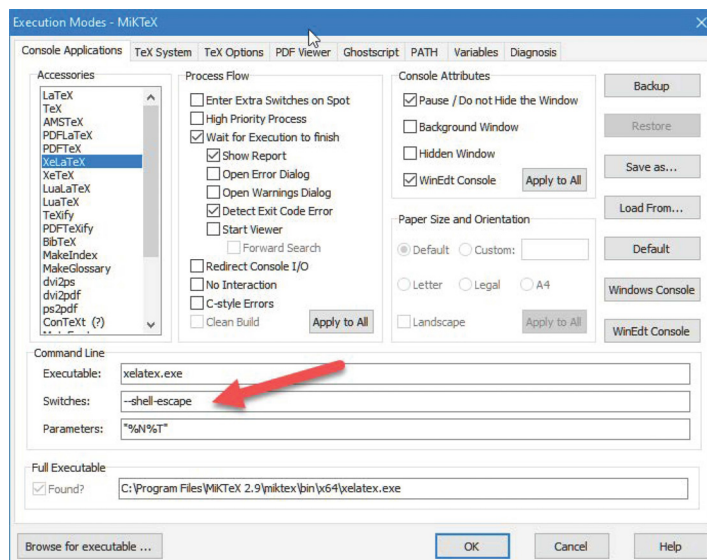


Figure 6: Setting `shell-escape` on WinEdt

2. **For all other cases of workflow.** When the icons file is compiled, the result is a *single* PDF (`icons.pdf`) with a page for each long question in the puzzle file, in an order determined by the randomization option (`!nonrandomized` or `nonrandomized`).

For a `xelatex` author who has Acrobat, the `useacrobat` needs to be used.

Additional comments for `xelatex` authors. In the case the author does not own Acrobat, when the icons file is compiled, each of the files `\jobname-sc(<num>).cut` is wrapped in a document template, compiled, and saved as `icon-<num>.pdf`. The document wrapper is determined by the contents of the `icondoc` environment. The default declaration of `icon-body` is,

icondoc env

```
\begin{icondoc}
\documentclass{article}
\usepackage{web}
\margins{3pt}{3pt}{3pt}{3pt}
\screensize{9\baselineskip}{2.25in}
\begin{document}
\small
\dpsInputContent      % required, defined in icon-doc
```

¹¹The name of the file is hard-wired into the package and cannot be changed at this time.

```
\end{document}
\end{icondoc}
```

Notice that this markup is similar to the source file for `icons.tex`; they differ in two respects, however: (1) the `icon-doc` package is not used; and (2) in the body, we use `\dpsInputContent` rather than `\createRequiredIcons`. If you use multiple workflows, the layout of the icons file and the declarations of the `icondoc` environment should be the same. Changes to the `icondoc` environment are made in the `icons` file. For example,

```
% icons.tex
% modify a design parameters
\documentclass{article}
\usepackage[!useacrobat]{icon-doc}
\margins{4pt}{4pt}{4pt}{4pt} % left,right,top,bottom (web command)
\screensize{10\baselineskip}{3in} % height,width (web command)
% change icondoc to reflect changes above
\begin{icondoc}
\documentclass{article}
\usepackage{web}
\margins{4pt}{4pt}{4pt}{4pt}
\screensize{10\baselineskip}{3in}
\begin{document}
\small
\dpsInputContent % required, defined in icon-doc
\end{document}
\end{icondoc}
\begin{document}
\small
\createRequiredIcons{12}{stat_match1}
\end{document}
```

Why is xelatex so special (such a problem)? The basic problem when dealing with xelatex is that it does not obey the `page` key of `\includegraphics`. When a graphic is embedded in a document we say,

```
\embedIcon[name=Q\n,hyopts={page=\n}]{icons.pdf}
```

where, `\n` is question number; in this case, `\n` is also the page number where the question is located in `icons.pdf`. Since, the key is not obeyed, we cannot bundle all the questions in a single PDF and pull out the page we want; no, the questions must be their own PDF, the embedding command is then,

```
\embedIcon[name=Q\n]{icons-\n.pdf}
```

where `\n` is the question number.

Hey, we're almost done with this option!

- **The workflow to build the puzzle file**

Let's summarize the workflow to build a puzzle file that uses icon appearances to display the long questions. We assume you have jumped through all the hoops of the previous section, your puzzle file and icons file are ready to go.

1. Compile the puzzle file, using the compiler of your choice. If you require a randomized listing of the questions and answers, either compile with `!nonrandomized`, or the `nonrandomize` option commented out or deleted. If you are randomizing, compile with the command `\inputRandomSeed` in the preamble to input that same initial seed back in when you later compile the puzzle file in step 3.¹²
2. Compile the icons file, using your favorite compiler. If you are a xelatex user, the option `useacrobat` and `!useacrobat` as appropriate. (Refer to **Additional comments for xelatex authors** on page 22 for more details.)
3. Return to the puzzle file. Compile the puzzle file again to obtain the final version.
4. Bring your PDF into Adobe Reader DC (or Adobe Acrobat DC if you have it) and save the file. After saving, test the puzzle to be sure the icons are displayed with the long questions. If it does not work, repeat steps 1-3 more carefully.

To familiarize yourself to the procedure, build the demo file `stat_match1.tex` using your favorite PDF creator. Try it with several PDF creators, just for fun. The process is straight forward once everything is properly set up.

9.2. The `uselayers` option

Demo files. There are several example files in the `examples/advanced/ocgs` folder, but in this discussion, we'll reference, once again, `stat_match1` found in the `ex1` folder. There are two versions of this file `stat_match1-ep.tex` and `stat_match1-tb.tex`. The latter uses the `textpos` package, the former uses the `eso-pic` package. For variety, we'll reference the `textpos` version.

The `uselayers` option is actually a simpler approach (no icon file needed), but there has more restrictions on the workflow. This option requires the `aeb_pro` package (with correctly installed JS files `aeb.js` and `aeb_pro.js`) and requires a `dvips->distiller` PDF creator workflow.

aeb_pro pkg

Acrobat required

- **The puzzle file**

A minimal specification for the `aeb_pro` and `dps` packages is listed below:

```
\usepackage[%
  web={extended,tight},
  eforms,
  uselayers
]{aeb_pro}
\usepackage[uselayers,
```

¹²Or, you can open the SAV file and copy and paste the initial seed into the argument of `\useRandomSeed`.


```

        nonrandomized,
        savedata
    ]{dps}

```

Posing long questions. The `\DeclarePuzzle` is as described in [Section 5.1](#). As in the case of [Section 9.1](#), the `setContent` environment is used to pose long questions.

```

\begin{cQ}{\langle name \rangle}
  \langle question-prompt \rangle
\begin{setContent}{\langle name \rangle}
  \langle long-question \rangle
\end{setContent}
\end{cQ}

```

The definition of `setContent` differs from that of `setContent` for the `usebtnappr` option. A long question is not required.

The `setContent` environment writes its content verbatim to a CUT file (named `\jobname-sc(\langle name \rangle-\langle num \rangle).cut`). For example, the CUT file for the first question of the demo file reads,

```

\textbf{Problem 1}\newline
The two branches of statistics
are descriptive and \underbar{\hspace{.5in}}.

```

The first line, which is not part of the `setContent` content, seen above can be modified using the following two commands.

```

\newcommand{\quesNumTxt}[1]{\protect\textbf{Problem #1}}
\newcommand{\quesNumTxTPost}{\protect\newline}

```

These may be redefined.

In the body of the puzzle page. The following three commands are placed on the same page as the puzzle is to appear. They precede the layout of the puzzle, questions, and answers.

```

\fmtOCGQues{\langle formatting-comm \rangle}
\placeQuesLayer{\langle placement-cmds \rangle}
\placeOtherLayer{\langle placement-cmds \rangle}

```

`\fmtOCGQues` The argument of `\fmtOCGQues` is a convenient way of designing how your long question appears on the page. Within the argument, use `\dpsQuesLayer#1` to symbolically reference the question. For example, from the demo file,

```

\fmtOCGQues{%
  \parbox[t][9\baselineskip][t]{2.25in}{\kern0pt\small\hfuzz11pt
  \psshadowbox[framesep=0pt]{\fcolorbox{red}{cornsilk}{%
  \parbox{\linewidth}{\dpsQuesLayer{#1}\vskip3pt}}}}
}

```

Here, we've used `\pshadowbox` from `pstricks-add`,

The nature of the `\placement-cmds` depends on the "placement" package used. The demo files use either the `eso-pic` or `textpos` package. The `\placement-cmds` argument places the question `\insertQuesLayer{#1}`.

*eso-pic, textpos
packages*
`\placeQuesLayer`

For the command `\placeQuesLayer` places the question layer, symbolically represented by `\insertQuesLayer{#1}`. For example, from the demo file,

```
\placeQuesLayer{%
  \begin{textblock*}{2.25in}[0,0]2.5in+.725in,3in
  \insertQuesLayer{#1}
  \end{textblock*}%
}
```

`\insertQuesLayer` sets the layer and inputs the formatted content.

`\placeOtherLayer`

Use `\placeOtherLayer` to place other non-question content in a layer. For example, from the demo file,

```
\placeOtherLayer{%
  \begin{textblock*}{2.25in}[0,0]2.5in+.725in,3in\centering
  \xBld{owclogo}\parbox{2.25in}
  {\includegraphics[width=2.25in]{owc_self}}\eBld
  \end{textblock*}%
}
```

Here, you must use the `aeb_pro` commands to create a named layer, using the `\xBld/\eBld` command pair. The argument of `\xBld` is the name of the layer. Within the `\xBld/\eBld` pair, we create our content of a graphic.

9.3. Developing an end of game event

There are commands and JavaScript hooks to enable a knowledgeable author to create a special end-of-game event. For example, the graphic placed by `\placeOtherIcon` or `\placeOtherLayer`, becomes visible when the player finishes the puzzle. All the demo files in the advanced folder have end of game events. The `dps` provides the command `\dpsFinishedEvent` to add JavaScript lines to the action to the end of game event. The command takes one argument to pass the JavaScript lines to the end of game event; for example, `\dpsFinishedEvent{dpsShowFld("btnEmoji");}` appears in several of the example file. This shows the `btnEmoji` image when the player has finished the puzzle.

\dpsFinishedEvent

10. Creating a sideshow

Some of the advanced examples use a sideshow; as the player progresses through the puzzle, with each success, a new piece of a graphic is revealed. A partially worked puzzle is shown in [Figure 7](#).

There are two commands, placed in the preamble, that control the behavior of the sideshow.

```
\randomizePicMappings
\sortPicMappings
```

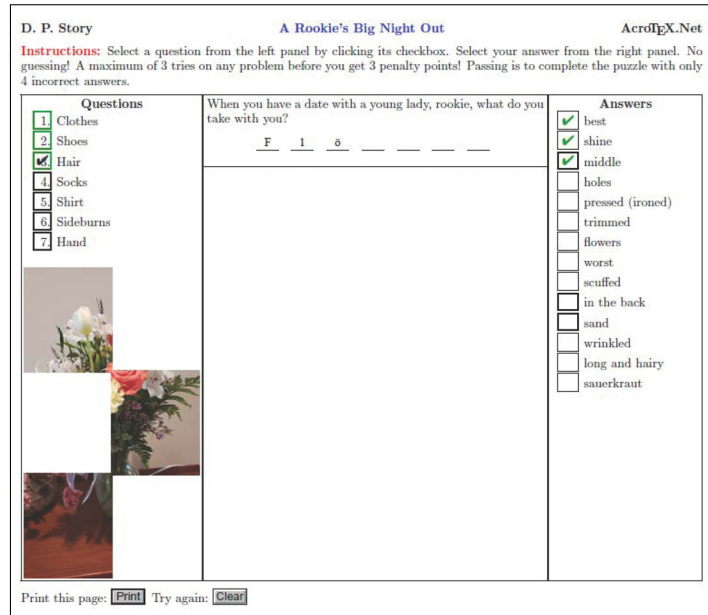


Figure 7: Puzzle with sideshow, shown on left

The order the sideshow pictures appears can be in their natural order, or in a randomized order. To randomize the order, insert `\randomizePicMappings` in the preamble; otherwise, the pictures appear in their natural order.

Second command, `\sortPicMappings` implies `\randomizePicMappings`, but with a twist. The pictures are placed in random positions as they appear; at the end of the puzzle, a bubble sort is applied, and the pictures are sorted to their natural order. Cool.

In all cases, pressing the Clear button creates a new randomization of the sideshow.

Tiled graphics. A sideshow appears in pieces, called *tiles*, that appear one at a time as the puzzle is solved, see Figure 7. The graphic to be used must have been broken down into a series of *tiled sub-graphics*. These tiles must be created in a certain way and labeled in a specific manner. Use the `tile-graphic` package to tile the graphic into either individual PDF tiled sub-graphics or as a single package of tiled graphics.¹³

tile-graphic pkg

The base name of the graphic. The `tile-graphic` has a naming convention that this package respects:

- When the tiled graphics are individual files, they are named, for example, `mypic_01`, `mypic_02`, `mypic_03`, The base name of this example is `mypic`. The graphical file format of the tiles is any format the PDF creator supports for graphical inclusion. It is usually most convenient for the tiled files to be PDF files.

¹³<https://ctan.org/pkg/tile-graphic>

- When the tiled graphics are packaged,¹⁴ the tile-graphic package names the package file, for example, `mypic_package.pdf`. When packaged, the graphical file format for the tiles are always PDFs. The base name of this example is `mypic`.

Additional details of how to create a sideshow are dependent on whether the option `usebnappr` or `uselayers` is taken.

10.1. Sideshow with the `usebnappr` option

Demo file. The demo file for this feature is `first_date.tex` found in the folder `examples/advanced/usebnappr/sideshow`.

First we embed the pictures of the sideshow in the embedding environment.

```
\begin{embedding}
\dpsEmbedIcons
<other-embedding commands>
\sideshowPackaged % optional
\dpsEmbedSideShow[<ext>]{<n-pics>}{<path>}
\end{embedding}
```

There are two ways to present the sideshow pictures, as described in the paragraph **Tiled graphics** above, to the `\dpsEmbedSideShow` command: (1) as individual tiled graphics; or (2) as a packaged graphic, the pages of which are the tiled graphics. The form of how the picture is presented is signaled to the `\dpsEmbedSideShow` command by the command `\sideshowPackage`, as indicated above. The first argument is `<ext>`, the file extension of the graphic, this usually not needed; when there is no `<ext>` specified, the extension is assumed to be `pdf`. The second argument is `<n-pics>`, the number of tile sub-graphics in the sideshow. The third argument, `<path>`, is the path to the sideshow graphic. At the end of the `<path>` is the base name of the graphic, as described in **The base name of the graphic** on page 27; for example, `graphics/mypic` indicates the tile files are in the `graphics` sub-folder, with base name of `mypic`.

After embedding the sideshow graphics, insert them into the puzzle board using,

```
\insertSideshow{<nRows>}{<nCols>}{<width>}{<height>}
```

where, `<nRows>` is the number of rows of the tiled graphic; `<nCols>` is the number of columns of the tiled graphic; `<width>` is the width of the tile; and `<height>` is the height of the tile. These latter two are adjusted so the tile fits into the space allotted and the aspect ratio is preserved.

Remark. When compiling a puzzle with the `usebnappr` option, the puzzle file compiles in two “modes,” depending on the state of the switch `\ifwrtContent`. Each time the document is compiled, the package looks for the file `icons-pglst.sav`, if it exists, the switch `\ifwrtContent` is set false; otherwise, it is set to true. The file `icons-pglst.sav` is created by `icons.tex`, the existence of `icons-pglst.sav` means `icons.tex` has been built. Once the puzzle file knows the `icons.tex` is built, the

¹⁴`xelatex` does not support packaged files, the tiles should be individual PDFs, as describe previously.

switch `\ifwrtContent` is set to false, and its behavior is changed slightly. If something goes wrong, delete `icons-pglst.sav`, and rebuild the puzzle file first, the icons file next, then finally the puzzle file again.

10.2. Sideshow with the `uselayers` option

Demo file. The two TEX files in the `examples/advanced/uselayers/sideshow` folder. When the `uselayers` option is specified, only EPS files are supported.

```
\insertSideshow{<nRows>}{<nCols>}[<hy-opts>]{<path>}
```

11. Some small degree of security

If a puzzle is create for a class of students to take for credit; then some security is appropriate. Typically, you post the puzzle with a print button (see `\printDPS` above. After the student completes the puzzle, he/she prints the results and hands it in for some credit. The `dps` package also provides `\clearOnCloseOrSave`:

```
\clearOnCloseOrSave
```

Place this command in the preamble. Now, when the puzzle is built, a student tries to close or save the puzzle, the puzzle is cleared before closing or saving. The only record is the printed version.¹⁵

12. Let's have some Fun

In order to make answering the questions “fun,” and in addition to the puzzle (or message), I implemented a point system. Each time the user checks an incorrect answer, that is recorded as a miss. After completion of the game, the JavaScript determines if the user has passed the test. To make it more interesting, a penalty point system is also used: If the user incorrectly answer the same questions multiple times (guessing!), penalty points are given.

The document author can set the various parameters of this aspect of the game.

`\threshold{<n>}`: The number, $\langle n \rangle$, of times a person is allowed to miss the same question before being “awarded” penalty points. The command `\threshold` with its one argument defines the command `\dsthreshold` which expands to the argument, n , of `\threshold`. Set `\threshold` in the preamble, and use `\dsthreshold` as part of the instructions or description of the game. The default: `\threshold{3}`.

`\penaltpoints{<n>}`: The number, $\langle n \rangle$, of penalty points to be added into the final score. Penalty points are “awarded” for missing the same question more than the number specified by the argument of `\threshold`. The command `\penaltpoints` with its one argument defines the command `\dspenaltpoints` which expands to the argument, n , of `\penaltpoints`. Set `\penaltpoints` in the preamble, and use

¹⁵Perhaps this is a mere nuisance, the student can make many copies of the printed puzzle, and hand it around to others in the class. Nothing is foolproof.

`\dspenaltypoints` as part of the instructions or description of the game. Default: `\penaltypoints{3}`.

`\passing{<n>}`: The maximum number, $\langle n \rangle$, of questions the user needs to miss and still pass the test. Passing or not does not depend on the number of penalty points. The command `\passing` with its one argument defines the command `\dspassing` which expands to the argument, $\langle n \rangle$, of `\passing`. Set `\passing` in the preamble, and use `\dspassing` as part of the instructions or description of the game. The default is `\passing{4}`.

The number of incorrect answers and the total penalty points are combined. Based on the combined score, a final evaluation of the user's knowledge on the subject is displayed.

13. Checking for validity

When creating the game, human error can sneak in. The most critical part is the `\DeclarePuzzle` command and getting the names of your fields set up the way you want. Letters with the same field name (the second parameter of the pair) will only need one question, they will all “light up” when the question is answered.

As explained earlier, after you've decided on your puzzle and the field names, you can then create your `Composing` environment using the `\writeComposingEnv` helper command. Review the discussion in ‘[Begin composing the questions and answers](#)’ on page 12.

To help you lay out your design, use the `viewmode` option, possibly along with the `showletter` option. This gives you a nice preview and you can see where everything is located. You may have to adjust the parameter for `\insertPuzzle` to fit the puzzle into the allotted space. If enclosing puzzle, questions and answer in frames, there may have to be some adjustment of the depth of the controlling `minipage` environments, and so on, etc., etc., etc., and, of course, etc.

Assuming you have successfully posed questions and answers, and designed a layout for your puzzle, the questions, and the answers, you are ready to test it. Using the `nonrandomized` option is nice for checking your puzzle; the questions and answers are listed in the same order.

If you have customized the text using the `lang=custom` option, you need to check that your text displays correctly; this is especially important if your new text contains accented characters, such as our old friend ü. To test your customized string, open your puzzle document in Acrobat (Reader will not do here) and execute these JavaScript lines in the console:¹⁶

```
this.resetForm()
nMissed = 0;
nPenaltyPoints = 0;
nPassing = 4;
checkForFinished();
```

¹⁶To execute from the console, open the console window by pressing `Ctrl+J`, paste in the code, highlight all the lines and press the `Enter` key on the keyboard, or `Ctrl+Enter` on the keypad.

By changing the variables `nMissed`, `nPenaltyPoints` and `nPassing`, and executing, you get the different messages appearing in the message text field.

14. Using the web Package

The web package has a many features that can be utilized as a part of your overall puzzle design.

The package has a powerful template management system for inserting background graphics into a document, and a system for painting the background a color other than the default white.

Use the `\margins` and `\screensize` commands to set the dimensions of your puzzle game page:

```
\margins{\left}{\right}{\top}{\bottom}
\screensize{\height}{\width}
```

Enter the title and author's name, as well as other metadata:


```
\title{\doc-title}
\author{\doc-author}
```

See the documentation, `aeb_man.pdf`, of the [AeB distribution](#) for details.

15. Thanks

My thanks to Jürgen Gilg, of u-umlaut fame, for his help and kind suggestions during the development of this game.

That's all for now. Hope you enjoy *Das Puzzle Spiel* and find it a useful learning tool.

Now, I simply must get back to my retirement. 

16. Appendix

The following is a subset of the PDFDocEncoding character set for PDF, these are useful for creating your custom localization file `dps_str_cust.def`, as discussed in 'Package Options' on page 6.

16.1. German Umlaut (dieresis)

Here a little tabular how to substitute the German Umlaut (dieresis) in PD1.

Ä	<code>\string\304</code>	Ö	<code>\string\326</code>	Ü	<code>\string\334</code>
ä	<code>\string\344</code>	ö	<code>\string\366</code>	ü	<code>\string\374</code>
ß	<code>\string\337</code>				

16.2. Accents

Here a little tabular how to substitute accents in PD1.

À	<code>\string\300</code>	È	<code>\string\310</code>	Ì	<code>\string\314</code>
à	<code>\string\340</code>	è	<code>\string\350</code>	ì	<code>\string\354</code>
Á	<code>\string\301</code>	É	<code>\string\311</code>	Í	<code>\string\315</code>
á	<code>\string\341</code>	é	<code>\string\351</code>	í	<code>\string\355</code>
Â	<code>\string\302</code>	Ê	<code>\string\312</code>	Î	<code>\string\316</code>
â	<code>\string\342</code>	ê	<code>\string\352</code>	î	<code>\string\356</code>
Ë	<code>\string\303</code>	Ë	<code>\string\313</code>	Ë	<code>\string\317</code>
ë	<code>\string\343</code>	ë	<code>\string\353</code>	ë	<code>\string\357</code>
Ò	<code>\string\322</code>	Û	<code>\string\331</code>	Ç	<code>\string\307</code>
ò	<code>\string\362</code>	ù	<code>\string\371</code>	ç	<code>\string\347</code>
Ó	<code>\string\323</code>	Ú	<code>\string\332</code>		
ó	<code>\string\363</code>	ú	<code>\string\372</code>		
Ô	<code>\string\324</code>	Û	<code>\string\333</code>		
ô	<code>\string\364</code>	û	<code>\string\373</code>		