JJ_Game Class

D. P. Story
# Table of Contents

1 Introduction ........................................ 3

2 Unpackaging the Class .............................. 3

3 Options .............................................. 3

4 Designing the Game .................................. 4
   4.1 Stuff for the Preamble .......................... 4
      • Banners and Backgrounds .................. 5
      • Game Design .................................. 5
   4.2 The Instructions ................................. 7
   4.3 The Questions .................................. 8

5 Building the Game .................................. 8
1. Introduction

The “JavaScript Jeopardy Game”, a.k.a., jj_game, is a \LaTeX{} class file that can be used to construct a Jeopardy-like game board and the accompanying answers and questions. The jj_game is capable of constructing game board for an arbitrary number of categories and an arbitrary number of question per category.

There are options for the user of \texttt{dvipsone} (from \TeX{}\&\LaTeX{}), \texttt{dvips}, \texttt{pdftex} and \texttt{dvipdfm}. There is also an option to use \texttt{foils}; this enables you to construct a game board that might be suitable for presentations, perhaps in the classroom.

- For people using \texttt{dvipsone} and \texttt{dvips} pay particular attention to ‘Building the Game’ on page 8, this section has special instructions for you.

- For the people who use \texttt{pdftex}, jj_game uses the \texttt{\pagecolor} command to produce background colors and requires v0.03f or later of \texttt{pdftex.def}, which is part of the standard \LaTeX{} command from the “Graphics Bundle”.

I do hope there will be people out in Cyberland who will use jj_game and construct some interesting games for us to enjoy.

Now, I simply must get back to work. \TeX{}

2. Unpackaging the Class

To unpack the jj_game.dtx, ‘latex’ the installation file jj_game.ins. If you use Acrobat 5.0, open the file jj_game.ins in your favorite editor and change the line

\begin{verbatim}
def\AcrobatOption{Acrobativ} % Acrobativ or Acrobatv
def\AcrobatOption{Acrobatv} % Acrobativ or Acrobatv
\end{verbatim}

then do a ‘latex jj_game.ins’.

When you distill with the Acrobat Distiller 5.0, the document-level JavaScripts needed to operate this game are automatically inserted when the game is opened in Acrobat for the first time. See the section ‘Building the Game’ on page 8 for more details.

3. Options

As if this writing, there are seven options.

- \texttt{\documentclass[dvipsone]{jj_game}}
  The dvi-to-ps converter by \TeX{}\&\LaTeX{}. In this case, the Adobe Acrobat Distiller must be used to convert from Postscript to PDF.
4. Designing the Game

The following sections outline out to design the game. If you don’t specify a particular part of the design of the game, then a default value will be substituted.

The sample file jj_test.tex pretty represents prototype game that you can modify.

4.1. Stuff for the Preamble

There is potentially a lot stuff that goes into the design of the game. Again, all have default values, but there are some required design parameters.
• Banners and Backgrounds
On the first page, where the instructions are usually placed is a horizontal rule with a game title in it. The text and colors are controlled by the following macros and values.

All the following can be redefined.

% Title of the game, that appears in the opening banner
\titleBanner{The \TeX\ Game!} % required!
% color for background and text on running banner
\definecolor{fillBanner}{rgb}{0.5,0,0.5}
\definecolor{textBanner}{rgb}{1,1,0} % yellow
\renewcommand\bannerTextFont{\sffamily\huge}

% color text the categories appear in game board
\definecolor{textBoard}{rgb}{1,1,0} % yellow

% background colors for different categories of pages
\definecolor{fillInstructions}{rgb}{0.98,0.92,0.73}
\definecolor{fillGameBoard}{rgb}{1,0.8,0.6}
\definecolor{fillQuestions}{rgb}{1,0.8,0.6}

% Background color for the cells, after a cell is chosen
\definecolor{fillCells}{rgb}{0.98,0.92,0.73}

• Game Design
\GameDesign: In order to design a game you must have categories and questions in each category. The \GameDesign is the macro that is used for this purpose.

\GameDesign
{
  Cat: \TeX, % list the categories
  Cat: \LaTeX, % 
  Cat: Classes \&~Packages, % 
  NumQuestions: 2, % number questions per category
  Goal: \{3,500\}, % set a goal, just for fun
  GoalPercentage: 0, % goal as a percentage of money
  CellWidth: 1in, % cell width
  CellHeight: .5in, % cell height
  ExtraWidth: 0pt, % additional width if needed
  ExtraHeight: .2in, % additional height if needed
}

The key-value pairs can appear in any order.

  Required Keys: The Cat and NumQuestions keys are required, though this is not enforced.

  The format is ‘<key>: <value>,’ there must be a space after the colon, and a comma following the value. If the value contains a comma, enclose the value in braces, as I did above with the Goal.
Each key, except Cat, has a default value. The values of these keys may need to be adjusted depending on the number of categories, number of questions, and font size.

The value of Cat has an optional argument, you can say

Cat: [\small] Classes & Packages,

The value of the optional argument is inserted just before the “Classes & Packages” in the category heading of the game board. In this case, I’ve reduced its size to fit into the bounding rectangle.

The parameter GoalPercentage needs comment. You can set the goal by setting the percentage of the total money in the game; i.e., GoalPercentage: 90, sets the goal to 90% of the total money. If GoalPercentage is present in the \GameDesign, and its value differs from the default value of zero (0), then the value of the Goal parameter, if present, is discarded, and a new goal is calculated based on the percentage of the total money, which is a function of the number of categories, the number of questions per category, and whether the double option has been invoked.

The jj_game takes these entries and computes the \paperheight, \paperwidth, \textheight and \paperwidth.

I’ve used some code from Radhakrishnan C V’s pdfscreen. In that regard, you can use marginsize to set the margins around the page, the default is

```
marginsize{.25in}{.25in}{.25in}{.25in}
```

There is a hidden banner above the game board that is revealed when the game is over (all questions are answered) and the goal, set in \GameDesign, is met. The \APHidden controls the appearance of this hidden banner.

```
\APHidden
{
Champion: You are a Champion!,% this text appears if goal met
Font: TiRo,
Size: 20,
TextColor: 0 0 1,
BorderColor: 0 0 0,
FillColor: 0.98 0.92 0.73,
}
```

\APDollar: The \DesignGame just sets basic parameters, it does nothing towards defining color. For that, I have the \APDollar macro:

```
\APDollar
{
Font: TiRo,  %# font to be use for the dollar amounts
```
The above are all the defaults. The \APDollar macro does not have to appear, unless you want to change one of the defaults. For example, you might want to say \APDollar{Size: 30}, in the case of the foils option.

\APRight and \APWrong: Under the face of the cells are two hidden text fields, one for a “Right” answer, one for a “Wrong” answer. The two macros \APRight and \APWrong have the same key-value arguments.

\APRight{
  Font: TiRo, Size: 20, TextColor: 0 0 1, Message: Right!}

\APWrong{
  Font: TiRo, Size: 20, TextColor: 1 0 0, Message: Wrong!}

\APSscore: There is a text field to keep track of the score. The following macro designs the score board.

\APSscore{
  Font: TiRo,
  Size: 20,
  TextColor: 0 0 1,
  BorderColor: 0 0 0,
  FillColor: 0.98 0.92 0.73,
  CellHeight: \the\cellHeight, % default same as cells
  CellWidth: \the\cellWidth, % default same as cells
  AutoPlacement: true, % auto placement of score
  Score: ",", % score text
  Currency: "$", % currency
}

Then AutoPlacement is true, the \ScoreBoard is centered directly under the game board. If false, then you have to provide a placement for the \ScoreBoard. Use the \PlaceScoreBoard macro:

\PlaceScoreBoard{\vfill\hspace{\rulewidth}\hspace{\extraWidth}\%\medskip\ScoreBoard}

This places the score board at the bottom of the page, aligned with the left edge of the game board.

4.2. The Instructions

The first page of the game is the instructions. Place the instructions in the instructions environment.
\begin{instructions}
<instructions go here>
......................
<perhaps a graphic as well>
\end{instructions}

The \end{instructions} ends the page, and inserts the game board on the next page.

### 4.3. The Questions

Following the instructions environment comes the question and multiple choice alternatives. For this, there is a \texttt{Category} environment, within which is placed a series of \texttt{Question} environments. The number of questions per category must be the same as declared as the value of \texttt{NumQuestions} in the \texttt{GameDesign} macro.

The format is as follows:

\begin{verbatim}
\begin{Category}{<category_name>}
\begin{Question}[<num_of_columns>]
<Text of the question>
\Ans0 ... % a wrong answer
\Ans1 ... % the right answer
\Ans0 ... % another wrong answer
\end{Question}
\end{Category}
\end{verbatim}

The number of categories must be the same as was listed in macro \texttt{GameDesign}.

The \texttt{<category_name>} is required; it is used to send messages to the log about whether you are listing the categories in the correct order. It doesn’t have to be exactly the same as was defined in \texttt{GameDesign}.

☞ The macro \texttt{\Ans} typesets the answers, it takes one argument. An argument of ‘0’, e.g., \texttt{\An0}, means that answer is wrong; an argument of ‘1’, e.g., \texttt{\An1}, means the answer is the correct one.

☞ The default behavior is a column listing of the answers. You can have a tabular format by specifying an optional number in the brackets, \texttt{<num_of_columns>}.

And that’s all there is to it!

### 5. Building the Game

For people using \texttt{pdftex} or \texttt{dvipdfm}, there is no special processing. Just ‘\texttt{pdflatex-it}’ or ‘\texttt{dvipdfm-it}’, and you are ready to play!
For **dvipsone** and **dvips**, some additional steps are necessary. The **jj_game** class uses Document Level JavaScripts to control the game as it progresses.

For **dvipsone** and **dvips**, the Document Level JavaScripts need to be inserted after distillation. The method of insertion depends on the version of **Acrobat** used.

▶ For users of **Acrobat 5.0**, the document-level JavaScript is automatically inserted when **Acrobat** is opened for the first time, provided the `.pdf` was saved back into the same folder as the source file. (It is in this folder that **TeX** has written the DLJS in the form of an FDF file.) **Note:** You must have unpacked **jj_game** with the **Acrobatv** option, see the section entitled ‘Unpackaging the Class’ on page 3 for a discussion.

Finally, do a ‘**Save As...**’ from **Acrobat** to save the DLJS that have just been inserted. You are now ready to play!

▶ For users of **Acrobat 4.0–4.05**, the file **jjg_dljs.pdf**, which comes with the distribution, can be used to introduce the JavaScripts. This file contains the JavaScripts in it already; all you have to do is to transfer them to the newly build game. Here are the steps for easily doing that:

1. The newly distilled game is open in **Acrobat**. Click on **Document > Insert Pages**, browse, and choose the PDF file **jjg_dljs.pdf**.

2. The file is now inserted. Now, click on **Document > Delete Pages** and delete the page you just inserted!

3. Do a **Save As**, reload the file, and you’re ready to play!

When the file is inserted, its Document Level JavaScripts become part of the combined file. When you delete that page again, the scripts stay. This is an easy way of insert scripts at the document level.