Creating (mathematical) jigsaw puzzles using TEX and friends

Julian Gilbey

Abstract

The jigsaw puzzles considered here are constructed from shapes such as triangles, squares and so on, with questions and answers written along their edges. The aim is to match them up correctly. Related puzzle varieties are card sorts and dominoes. We describe a TEX- and Python-based system designed to author such puzzles. The input is a text-based YAML file; the output can include both printable PDFs for cutting up and Markdown files for potential conversion to HTML.

1 Background

1.1 History and benefits of (mathematical) jigsaw puzzle software

In 2005, Hermitech Laboratory created *Jigsaw 2005*, software designed to create mathematical jigsaws. These are puzzles consisting of square and/or triangular pieces which fit together to make a hexagon or other shape. Two edges match if a question on one matches the answer on the other; figure 1 shows an example of a completed jigsaw. These can be used within classrooms to help make learning more engaging and enjoyable. For example, some students feel disengaged by having to continuously write in mathematics lessons. Students may well attempt many more questions than normal and participate more willingly when solving a jigsaw puzzle than when answering textbook questions. It is also different from working with a computer-based activity in that it is more physical and can easily involve an element of collaboration with classmates.

Jigsaws can also be used to develop certain logical thinking skills. For example, some questions could have the same answer, so that students would have to realise this and determine which one of the possible pieces fits all of the constraints. There can also be blanks or '?' symbols used to indicate that an answer has been left out. Deliberate mistakes could be introduced to raise the level of challenge. Finally, the edges could be used to introduce distractors (as in the example shown), or they could be left blank, or they could be used to spell out a relevant word, phrase or sentence.

While the *Jigsaw 2005* software was originally designed for use in the mathematics classroom, the same puzzle structure could easily be used to assist in learning languages, scientific facts, and so on.

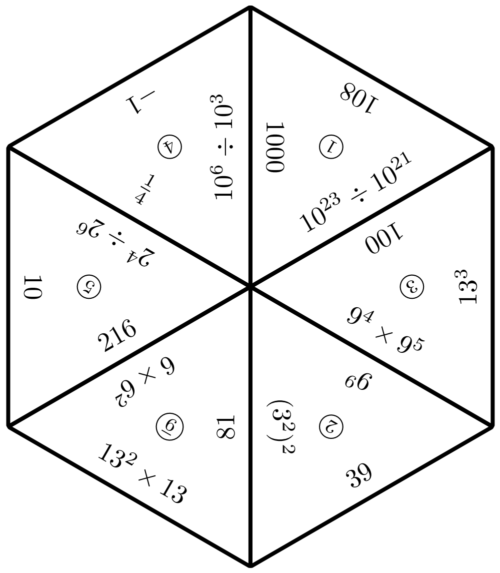

Figure 1: A small completed jigsaw

In addition to jigsaws, the software also offered dominoes, where each card has an answer and another question; these then match up to create a domino chain. Another type is a card sort, where a set of cards is to be sorted either into order (for example, a proof of the irrationality of $\sqrt{2}$ or the derivation of the formula for solving quadratic equations), or into groups (for example, "Which of these statements are always true, which are sometimes true, and which are never true?").

The *Jigsaw 2005* software was distributed to all UK schools and colleges with 16–18 year-old students as part of a project to improve the quality of mathematics learning. Since then, the software has been further developed, and is now freely available as *Formulator Tarsia* [3] (though it is not open source). Hundreds of classroom activities have now been written using this software.

The *Formulator Tarsia* software is Windows-based, and every question–answer pair is entered on a separate input screen. There are keyboard shortcuts available, but even having learnt them all, there is still a fair amount of mouse-work required to enter the questions. One also has to switch to different 'tabs' to see an overview of all entered pairs. On the other hand, the software learning-curve is very gentle, which is a huge bonus for busy teachers: though it may take time to create each jigsaw, there is very little up-front time investment required. Also, the data storage format is XML, with MathML for the mathematics, so it is potentially possible to edit the files outside of the software or to export the data.

1.2 The desire for a TeX-based system

I am currently developing mathematical resources for the Cambridge Mathematics Education Project (CMEP) [2]. We are aiming to provide innovative resources to help support and inspire teachers of advanced mathematics for 16–18 year-old students. In the UK, most of these students will study an A-level mathematics curriculum, but the material we are producing can certainly be used in other teaching situations and for other curricula.

Our build system involves authors writing the original content in Markdown, which is then converted using `pandoc` [4] to HTML for viewing and to LaTeX for producing a printable PDF document. (There are more steps in addition, but this lies at the heart of the conversion process.)

Among the resources we are developing are a variety of card-sorting activities, and there is a possibility that we may also offer some jigsaw activities. The *Tarsia Formulator* software does not meet our needs for a variety of reasons:

- All of our developers use either Mac OS X or GNU/Linux, so it would be hard for us to use Windows-based software.
- The resources are going to be made available via the web, so we require a way of having the content of our cardsorts available both for easy printing and for viewing online. It would be painful to have to enter the content from scratch into both a card-sort creator and a Markdown document.
- The WYSIWYG input system is quite laborious.
- We like the flexibility of LaTeX's mathematical typesetting abilities (in spite of the limitations imposed by using `pandoc` and *MathJax* [5]).
- It is desirable to have more fine-grained control over the output, should that be desired; without access to the source code, it is very hard to make any systematic changes to *Formulator Tarsia*'s output.
- When an activity has been created in *Formulator Tarsia*, it seems to be impossible to change the activity type (say from a triangle-based jigsaw to a domino puzzle) without either editing XML files by hand or re-entering all of the data; it would be much nicer to have a text-based input file which can simply have its header information modified appropriately or the content cut-and-pasted into another document.
- The output quality of the various TeX engines is superior.

For all of these reasons, I decided to create a TeX-based solution to address our needs and produce beautiful resources.

2 The new software

2.1 Design goals

My aim in creating this new software was to build on the great work that the creators of *Formulator Tarsia* had done, while addressing some of the shortcomings that we had identified above. The audience of this new software is also crucially different, in that it assumes its users have both a working knowledge of LaTeX and a TeX system installed. They will also need to have Python 3 and be able to install some standard Python modules. (I might endeavour to make the software compatible with Python 2.7 at a later stage if I have time and people express an interest in this.)

The primary design goals were:

- The content should be easy to input using a text editor (assuming they have a basic knowledge of LaTeX).
- The puzzle layouts should be editable if desired.
- New puzzle types should be easy to create, ideally without having to modify the source code.
- It should be easy to create the printable jigsaws from the input files.
- There should be sensible defaults which can be overridden if desired.
- It must be possible to include images in the puzzles.
- The software must be able to output a Markdown version of the puzzle content for including into our CMEP build process. The precise format of this must also be customisable.
- The software must be able to run on at least Mac OS X and GNU/Linux, and ideally on Windows too.
- It should be easy to install and use.
- The dependencies should be kept to a minimum (beyond a working LaTeX system).
- It should be easy to maintain (time is always in short supply).
- It should be possible for the jigsaw pieces to be automatically numbered.

For the last item, I often found as a teacher that it was very time-consuming to check whether a jigsaw puzzle or card sorting activity had been completed correctly. I would therefore number the pieces before photocopying a puzzle, to make this task more efficient. However, I still had to go to the effort of identifying the pieces in the solution to determine where the numbers would end up. So the ability to automatically number the pieces, both in the puzzle and solution, is very desirable. To make this effective, though, the cards had to be shuffled randomly for each different puzzle, so as to prevent the students from 'solving' a puzzle just by arranging the card numbers in some standard order.

A possible additional feature would be to create some form of graphical data input system, but I have no immediate plans to do so.

2.2 Implementation overview

The most important files from a user's perspective are the data input files. These specify the type of puzzle (a hexagonal jigsaw or a card sort, for example), any options for the jigsaw (such as whether to number the cards or not) and the text to appear on the puzzle. Sensible defaults are provided for all of the options if they are not specified. We decided fairly early on that YAML was a suitable markup language for writing these files, as it is a very easy format for humans to read and write, with little "noise". It also made more sense to use a standard, well-known markup language than to create a new one specifically for this project: it will then be easy for other software to read the input files or to change the files to a different format (such as JSON) should this ever be desired.

The puzzle templates are LaTeX files with templating marks. For the templates I have created, I have used PGF/Ti*k*Z for the graphics and to place the text items. My initial templates were written when I was using version 2.10 of PGF/Ti*k*Z, as distributed with TeX Live 2013. However, it turned out that I ran into a bug related to the placement of text along cyclic paths, which has been fixed in version 3.0.0 (distributed in TeX Live 2014). It is therefore necessary to have an up-to-date TeX Live distribution (or at least an up-to-date PGF) for these templates to work correctly. Alternative templates can be written if this is desired. For example, someone might prefer to use Asymptote or another graphical package, or they may wish to modify the existing templates in various ways.

There is also a template description file (again written in YAML) for each puzzle type: this specifies various parameters required to create the puzzle.

Both of these template types (the puzzle templates and the description files) are described in detail in the software documentation.

I wrote the program itself in Python. This was for a few reasons. Firstly, it is a well-known, popular language, so if other people wish to become involved in developing this software, it will be relatively easy for them to do so. Secondly, the Python interpreter is easy to install on the major platforms people will be using. Furthermore, since the software is written in pure Python, it does not require compilation, making things a little simpler to install. Finally, it provided me with an opportunity to learn more about this language: as I have learnt more about Python, I have improved my code and made it more idiomatic. The code is currently written in Python 3.x; if I have time and people express an interest, I will endeavour to make it compatible with Python 2.7.

```
type: smallhexagon
title: An example puzzle
note: 'You will have to work out the
  missing number shown as `?'''

pairs:
  - ['$10^6\div10^3$', '$1000$']
  - ['$10^{23}\div10^{21}$', '$100$']
  - ['$9^4\times9^5$', '$9^9$']
  - - '$(3^2)^2$'
    - puzzletext: '?'
      solutiontext: '$81$'
  - ['$6\times6^2$', '$216$']
  - ['$2^4\div2^6$', '$\dfrac{1}{4}$']
edges:
  - '$-1$'
  - '$10$'
  - '$13^2\times13$'
  - '$39$'
  - '$13^3$'
  - '$108$'
```

Figure 2: Example small hexagon puzzle

To create the jigsaws, the jigsaw generator program is run over the data file. It reads this file along with the relevant template description file and puzzle template files. It then fills in the puzzle templates with the puzzle data to create LaTeX files (and Markdown files too, if requested). The LaTeX files are then processed with pdfLaTeX (or some other LaTeX variant) to create PDFs for printing.

2.3 An example data file

Figure 2 shows the YAML puzzle file which was used to create the example shown in figure 1 above (with some small modifications).

The file begins with some metadata:

- the type of the puzzle: in this case, it is a 'smallhexagon' puzzle, which consists of six triangles;
- the title of the puzzle, which is optional, and
- an optional note, which is printed above the puzzle.

The existing jigsaw types at the time of writing are `hexagon`, `smallhexagon`, `triangle` and `parquet`; there are a few more in the pipeline, too. It is also possible to write one's own jigsaw types, as long as they consist of equilateral triangles and squares. (To write templates using different shapes would require extensions to the software itself; I may do this in the future.) In addition, there are three more types, `cards`, `cardsort` and `dominoes`, which are described briefly later.

The next section of the file specifies the data. There are two parts here: the pairs data, which lists question and answer pairs, and the edges data, which specifies the text to appear on the edges in an anticlockwise direction. Each pair is a sequence of two items, the question and the answer, whereas each edge consists of just a single item.

Each item is usually just a single string. However, there is a possibility of 'hiding' text in the puzzle. This means either leaving a blank where the text should appear or writing something else in its place. The fourth question–answer pair in the above example illustrates this. This pair is written using YAML 'block style' for clarity, that is, the question and answer are written on separate equally-indented lines. The question is just a plain string (`'$(3^2)^2$'`), whereas the answer is a mapping with two entries: the `puzzletext` appears in the puzzle, while the `solutiontext` appears in the solution. This feature could be used as in the example shown, or it could be used to introduce deliberate mistakes in the puzzle, increasing the level of difficulty for the students. There is also an alternative notation available to simply hide the text in the puzzle, which is described in the documentation.

If a puzzle does include such hidden text, then the solution highlights these occurrences; in the default templates, these are shown with a yellow background.

It is also possible to include images in the items or to change the text size of individual items or all items. The details are described in the documentation.

2.3.1 Some notes on YAML syntax

For a full, precise description of YAML syntax, see the YAML Specification [1]. What follows is a very brief summary of some of the basic parts of the specification which should be sufficient for most people's needs when using this software.

Although YAML allows unquoted strings in general, there are a number of restrictions on what is permitted in them. For this reason, it may be simplest to single-quote all value strings. (YAML also offers double-quoted strings, but these interpolate backslash-escapes; this is probably undesirable in this context, since many TeX expressions include backslashes.)

Within a single-quoted string, a single quote is written as a doubled quote mark (`''`), hence the three quote marks in a row at the end of the note field in the above example (two to indicate a quote, and the third to end the string).

For collections ('sequences' and 'mappings' in YAML's terminology, each of which consists of a number of 'entries'), YAML allows two different notations: either a flow style, which is similar to JSON notation, or a block style. With the flow style, the entries of a sequence, such as a question–answer pair or the list of edges, are enclosed in square brackets and separated by commas; for the block style, each entry appears on a new line preceded by a vertically-aligned hyphen.

Similarly, for a mapping the entries consist of key–value pairs, with the key and value separated by a colon. They can be written either using a flow style as a comma-separated list enclosed in braces, or with a block style by writing each key–value pair on an identically-indented new line.

Both of the sequence styles appear in the example here, though only the block style is used for mappings. Note also that the top-level structure of the file is itself a mapping. This means that the entries (type, title, pairs, and so on) can appear in any order. However, for the benefit of the human reader, it is wise to maintain a meaningful order to these entries.

2.4 Card sorts and dominoes

As mentioned above, this software also offers some other types of activities: card sorts and dominoes.

A domino puzzle is just a collection of question–answer pairs which are laid out on a series of dominoes. Each domino consists of an answer and a new question. The dominoes can then be laid out to create a complete chain (beginning with 'Start' and ending with 'Finish') or loop (if 'Start' and 'Finish' are not present), with each question matching its corresponding answer. The data file used to create this is very similar to the example shown above, only the type is now `dominoes`. Within the data file, it is also possible to specify various options such as how many dominoes should appear on a page and whether the cards should form a loop or chain; these are described in more detail in the documentation.

A card sort is simply a collection of cards which are to be sorted in some way. For the `cardsort` type, the aim is to sort the cards into the correct order, and so the cards are shuffled for the puzzle and a solution is also produced by default. For the `cards` type, on the other hand, there is no canonical order (for example: “Arrange these cards into order of importance to you” in a politics lesson), and so no solution is produced, nor are the cards shuffled by default. Again, the data file is very similar in structure, and details can be found in the documentation.

Both cards and dominoes offer the possibility of having some form of title on the cards, and cards have a further option of having labels on individual cards. (This was introduced into the software when we wanted to create an activity which had different categories of cards; the categories were then written on the cards.) Additional options are discussed in the documentation.

2.5 Markdown output

As explained in section 1.2, our requirements include the need to output a Markdown version of cards data for inclusion into our build system; from there, it is translated into HTML. Our current system requires each card to be embedded in an HTML `<div>` element. The cards are then displayed in an appropriate way using some simple CSS. Since `pandoc` passes any HTML `<div>` elements in a Markdown file to the HTML output unchanged, we simply need a Markdown file with the `<div>` elements already present for our card sorts. I have therefore created a Markdown template file which places the card content within these elements.

For other needs, it is perfectly straightforward to create alternative Markdown templates, which could then be converted into the required HTML. For example, with the appropriate JavaScript and supporting CSS, it would be entirely feasible to create an interactive version of the card sort or jigsaw from the same data file. While we have not yet done this, it would be a very interesting next step.

3 Status of the software

As the time of writing, the software is in alpha state. It is currently able to produce jigsaws and card sorts in all of the ways discussed in this article. There are a few key issues outstanding, which should be resolved in the very near future, including:

- creation of an installable Python package;
- handling command-line options;
- the ability to read a configuration file, specifying such things as the flavour of LaTeX to use;
- offering the ability to read user-defined template files, and
- writing the full user documentation.

Once these are done, I will consider the software to be in either beta or release-ready state. I welcome feedback and any suggestions for improvements or enhancements, as well as stories of how it has been used.

3.1 Obtaining the software

The software can be downloaded from GitHub; the repository is `https://github.com/juliangilbey/jigsaw-generator`. At the time of writing, the simplest way to obtain it is to use git to clone the repository. As mentioned above, I intend there to be an installable Python package by the time this article is published. It might then be appropriate to upload this package to CTAN. Information about this will be posted on the GitHub site.

Acknowledgements

I thank those who offered very useful ideas and feedback at the TeX Users Group conference; these have helped me to solve some of the thorny issues I had been facing.

References

[1] Oren Ben-Kiki, Clark Evans, and Ingy döt Net. YAML Specification. Available from `http://www.yaml.org/spec/1.2/spec.html`.

[2] Cambridge Mathematics Education Project. `http://www.maths.cam.ac.uk/cmep/`.

[3] Hermitech Laboratory. Formulator Tarsia. Available from `http://www.mmlsoft.com/index.php/products/tarsia`.

[4] John MacFarlane. The Pandoc universal document converter. Available from `http://johnmacfarlane.net/pandoc/index.html`.

[5] MathJax Consortium. MathJax. Available from `http://www.mathjax.org/`.

◇ Julian Gilbey
Department of Pure Mathematics and Mathematical Statistics
University of Cambridge
Wilberforce Road
Cambridge CB3 0WB
England
`J.Gilbey (at) maths dot cam dot ac dot uk`
`jdg (at) debian dot org`