level2image: A Utility for Making 2D Tile-Based Level Images with Overlays

Seth Cooper

Khoury College of Computer Sciences, Northeastern University
se.cooper@northeastern.edu

Abstract

2D tile-based levels are a common format for video game research, particularly in procedural content generation. Often, tiles are represented as text characters. Here, we describe level2image, a utility that provides a flexible means for converting such levels into a variety of image formats. It can use text tiles or substitute image tiles or backgrounds. There is support for various geometry overlays, such as paths through the level, areas of interest, or boundaries. The utility is a Python script, intended to reduce duplicated work creating such converters within the game research community.

Introduction

Video game levels that are 2D and tile-based are a common format for research in video games, particularly in procedural content generation (Shaker, Togelius, and Nelson 2016). Tiles are often represented as text characters in a text file, where, for example, an `X` character might represent a block and a `-` might represent empty space. This format is used, for example, by general projects like the Video Game Level Corpus (Summerville et al. 2016) and the Video Game Description Language (Schaul 2013), as well as projects for specific games like the Boxoban level set (Guez et al. 2018).

Often these text levels need to be converted to images, such as for figures in papers. It is also common for individual text characters to map to tile images (e.g. sprites), such as `X` mapping to  or `-` mapping to  . This mapping can be used in converting to level images.

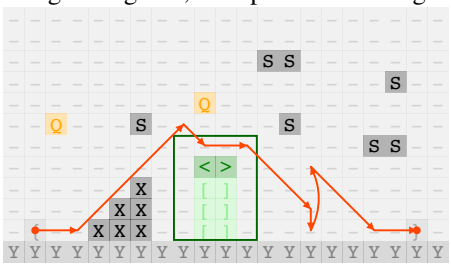
The `level2image` utility provides a flexible means for converting such text levels into a variety of image formats, using text or image tiles, and with geometry overlays. The utility is a Python script, intended to reduce duplicated work creating such converters within the game research community. The code is available on GitHub at https://github.com/crowdgames/level2image, under the MIT licence. Instructions for running the script are provided in the README. As the code is publicly available, and we are using it in our own work, we plan to maintain and update it.

`level2image` was originally created to overlay paths through levels generated by the Sturgeon constraint-based level generator (Cooper 2022). We expect the utility could be most useful in these cases: when levels are generated or processed, and additional metadata such as paths, areas of interest, or tiles to highlight can be stored with them and overlaid on the produced image.

Input text level and metadata

```
--------------------
--------------------
-----------SS-------
-----------------S--
---------Q----------
--Q---S------S-------
-----------------SS--
---------<>---------
------X--[]---------
-----XX--[]---------
-{--XXX--[]--------}-
YYYYYYYYYYYYYYYYYYYY
META {"type": "geom", "shape": "rect",
  "group": "misc", "data": [[6, 8, 11, 12]]}
META {"type": "geom", "shape": "path",
  "group": "path", "data": [[10, 1], [10, 3],
  [5, 8], [6, 9], [6, 11], [9, 14], [10, 14],
  [7, 14], [10, 17], [10, 19]]}
```

Image using text, with path and rectangle



Input image tile mapping

Image using image tiles

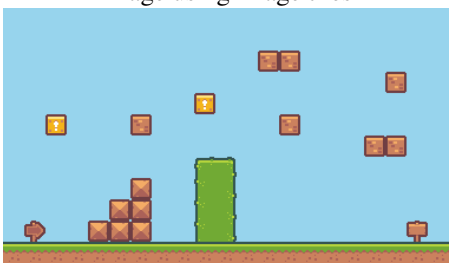


Figure 1: Level for a platform game and different produced images. A text to image mapping was used to make the bottom image.

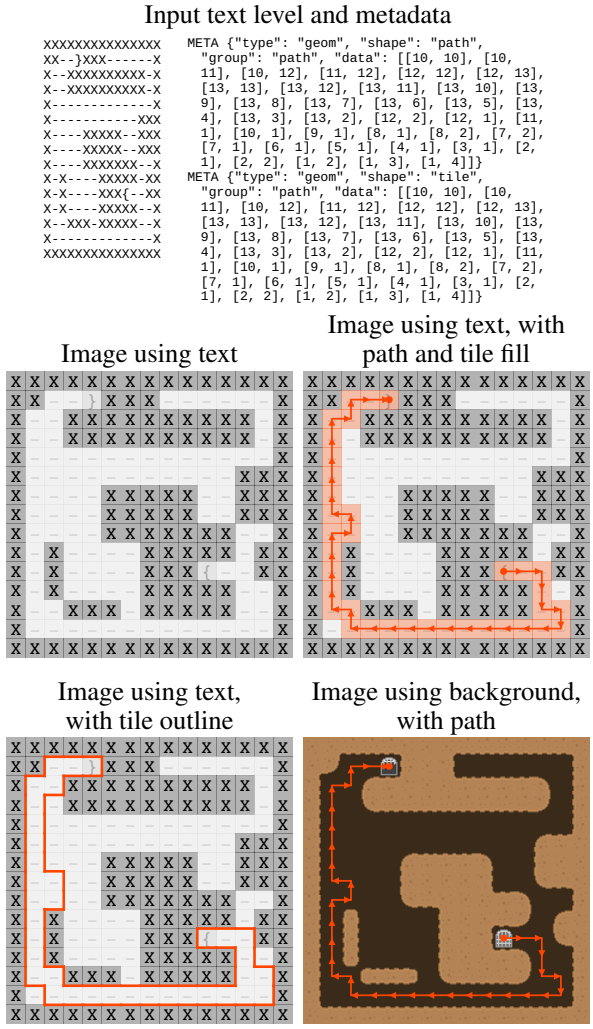


Figure 2: Level for a cave exploration game and different produced images. A single background image was used to make the bottom-right image.

Description

Although the technical details may change in the future, here we describe some of the current status of the `level2image` utility. Example text levels and various produced images are shown in Figures 1 and 2. Tile images used are from Kenney (Kenney 2022).

The utility takes as input, at a minimum, a text level file, which is a grid of text characters as is often used, with each character representing a tile. We extend the standard text level file format to also support metadata, specified by lines starting with `META`.

To create images, the utility internally creates an svg and then converts this to the desired format. There is currently support for outputting svg, pdf, and png image formats. Pdf is useful as it is a vector format and thus can scale nicely when embedded in another document as a figure. Sequences of images can also be output as animated gifs.

By default, the utility produces an image using the text character from each tile, which can be configured to use different colors for each character. It can also use a mapping from text tiles to image tiles, or substitute a complete background image.

There is support for various vector overlays in the images produced. Overlay geometry is currently specified by additional lines in the level file starting with the text `META` and followed by a JSON object string with the details of the metadata. Each piece of geometry can have a shape. These currently include: *paths*, meant to represent a directed path between two locations; *lines*, general line segments; *tiles*, tile locations; and *rects*, rectangles. Line-based geometry can curve, to make some attempt to prevent overlapping parts along paths (show in Figure 1).

Geometry can have visual styles applied to it for overlays, and can be organized into groups, which each have their own visual style applied. Groups can also be hidden or shown. Styles for line-based geometry includes arrowheads, dashes, thickness, and transparency. Styles for rectangle-based geometry include fill or outline, and tiles can additionally be styled to outline only the border around the tiles (i.e. lines between neighboring tiles are not shown). Colors for text tiles and geometry group overlays are specified by a configuration file, with a default configuration provided.

Conclusion

In this work we have presented the utility `level2image`, which creates images from a commonly-used text level representation. It supports geometry overlays for level annotation. We hope the utility will be useful in cases such as making figures for documents, and we expect it to evolve in the future as needs change.

References

Cooper, S. 2022. Sturgeon: tile-based procedural level generation via learned and designed constraints. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 18(1): 26–36.

Guez, A.; Mirza, M.; Gregor, K.; Kabra, R.; Racaniere, S.; Weber, T.; Raposo, D.; Santoro, A.; Orseau, L.; Eccles, T.; Wayne, G.; Silver, D.; Lillicrap, T.; and Valdes, V. 2018. An investigation of Model-free planning: boxoban levels. https://github.com/deepmind/boxoban-levels/.

Kenney. 2022. Free game assets. https://www.kenney.nl/assets. Accessed: 2022-01-07.

Schaul, T. 2013. A video game description language for model-based or interactive learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 1–8.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural Content Generation in Games*. Springer International Publishing.

Summerville, A. J.; Snodgrass, S.; Mateas, M.; and Ontañón, S. 2016. The VGLC: The Video Game Level Corpus. In *Proceedings of the 7th Workshop on Procedural Content Generation*.