Creating 2D and 3D Scenarios

Last modified: July 29, 2020

Python Scripts

The script (“convert.py”) is capable of creating 3D scenarios from 2D scenarios, 2D scenarios from image files, and 3D scenarios from image files. Which functionality is used depends on the input file. If the file extension is “bmp”, “jpg”, “jpeg”, or “png”, the script will attempt to convert the image to a 3D scenario. If the file extension is “json”, the script will attempt to convert the 2D scenario to a 3D scenario. If the input file’s extension is not any of the aforementioned options, the script will be unable to run.

If dimensions are provided as a command line argument (see the section on usage), the script internally converts the given image into a 2D scenario. It will then scale the scenario down to match the requested dimensions. The script will then either output the 2D scenario or convert it into a 3D scenario. Note that the scaling process is inexact.

Setting the *height* value in the configuration file to 1 will cause the script to produce a 2D scenario.

Configuration

The configuration file contains three sections: files, model, and image.

The *files* section determines where the input scenario/image is located and where the resulting JSON will be saved.

The *model* section provides details about how the model will be constructed. Importantly, much of the information in this section pertains to how the model will be converted from 2D to 3D. The *height* parameter sets how many cells tall the model will be. A height of 14 means that the model will be 14 cells tall. Since each cell is $25cm \times 25cm \times 25cm$, this corresponds to a total height of 3.5 metres. However, the room itself will be 12 cells (3 metres) tall as the floor and ceiling are included in the height value. Setting this value to be 1 will cause the script to produce a 2D model. The *walls_only* value, if set to true, will ignore all elements that are not of type -300 (IMPERMEABLE_STRUCTURE). Typically, this value should be false. The *heights* object sets where the specified elements will be placed on the new axis. For example, *door_top* sets highest cell that the door will extend to. Note that the highest possible value for these parameters is $height - 1$ since the first layer is at $z = 0$.

Within the *model* section, there is a subsection called *counter*. This area determines the *counter* value of each WORKSTATION (type -700) cell. The counter attribute of a cell determines when a WORKSTATION cell will become a CO2_SOURCE (type -200). The *seed* value is the seed for the random number generator which is used to set each cell’s counter value. The *minimum* and *maximum* fields specify the minimum and the maximum possible values for the counter value (both inclusive), respectively.

The *image* section in the configuration file contains information on how the script will convert different colours, the colour tolerance (how different a pixel in the image may be from a solid colour), and the alpha tolerance (how transparent a pixel can be before being turned into an AIR cell).

The colour tolerance is used to ensure that each component of each pixel’s RGB value is within the tolerance value of either 0 or 255. If the value is within the tolerance, the script will modify the pixel to be a minimum or maximum value (0 or 255). If the value falls outside of this tolerance, the script will attempt to resolve the issue by rounding the colour component to the closest of 0 and 255 and will display a message indicating that this occurred (if the “img-msg” flag is set). A higher colour tolerance allows for more variation from the colour scheme but may cause inadvertent cell types to be set.

The alpha tolerance is used to ensure that each pixel is opaque enough to be assigned a cell value based on its colour. If a cells alpha value falls below the alpha tolerance level, the cell will be considered an AIR cell, regardless of its colour. A higher alpha tolerance means that a cell is required to be more opaque to have its colour considered. Alternatively, a lower alpha tolerance means that a cell can be more transparent and still be assigned a cell type based on its colour.

The current default colours are as follows:

Colour	String	Cell Type	Type Number
white	255,255,255	AIR	-100
cyan	0,255,255	CO2_SOURCE	-200
black	0,0,0	IMPERMEABLE_STRUCTURE	-300
green	0,255,0	DOOR	-400
yellow	255,255,0	WINDOW	-500
blue	0,0,255	VENTILATION	-600
red	255,0,0	WORKSTATION	-700

Note that certain elements (such as the concentration increase for CO2_SOURCE cells or the default cell properties) cannot be modified in the configuration file. These properties are hardcoded into the script.

Usage

The script uses a Python module called Pillow (imported as “PIL” in the script) for image analysis. This module requires separate installation which can be done using pip. The module can be installed with the following command for Python 3:

```
pip3 install Pillow
```

The script itself has the following usage:

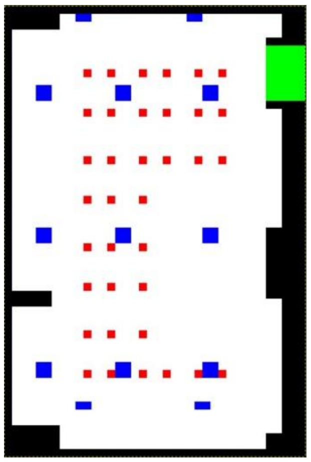
```
convert.py [-h] [--dimensions int int] [--progress-msg] [--img-msg] [--no-crit-msg] config
```

The arguments are as follows:

Flag	Description
config	Required argument; the path to the configuration file
--help, -h	Displays a description of the program and usage information
--dimensions, -d	Accepts two integers which represent the desired output dimensions; only used for image-to-JSON conversions
--progress-msg, -p	Enables messages which detail the progress of the conversion
--img-msg, -i	Enables messages which detail errors and notes that may occur that relate to image processing
--no-crit-msg, -c	Disables critical messages (such as file location and scaling issues); these messages show when an error causes the program to prematurely terminate

Creating Images

The script can work on a per-pixel basis. This means that each pixel is considered to be a cell. For a model with the shape $25 \times 30 \times 14$, the image should be 25 pixels by 30 pixels. Using an image manipulation program such as GIMP allows for the necessary amount of control. Programs such as Microsoft Paint also work but have a limited zoom and have no ruler.



It is important to note that an uncompressed image format should be used (such as a bitmap) for exact image processing. Using compressed formats (JPG, etc.) can result in pixels changing colours and blurring.

If the image is larger than the desired scenario, the user can specify the required dimensions on the command line. The script will then scale down the resulting scenario. This method is an inexact process which can result in strange results (such as thick walls or missing elements).

The picture to the right is an example of a model before being converted. It uses the default colour values.