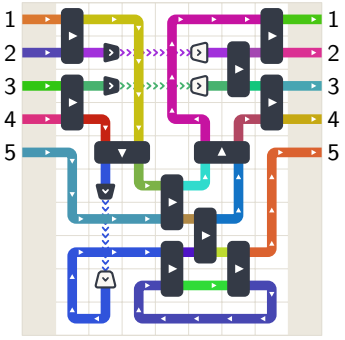
beltmap: schematic pictures of Factorio belt layouts

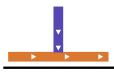
version 0.1.1, 2026-11-21

beltmap draws belt balancers and other belt-and-splitter layouts in LuaLaTeX. It reads Factorio-SAT tiles, search records, and Factorio blueprint strings of version 1.1 and 2.0, or a layout typed as a character grid, traces every belt from splitter to splitter, and draws the result in TikZ under one of five themes.

```
\usepackage{beltmap}
...
\beltlayout[labels]{examples/f00_9x10.json}
```



1 Reading a picture

Glyph	Meaning
	A belt. Every belt between two splitters has its own color, through turns and tunnels. Arrowheads give the direction.
	A splitter over two tiles; the arrow is its direction.
	Underground entrance: a solid hood over the half of the tile that faces the tunnel, narrowing toward it.
	Underground exit: the same hood, open, the belt leaving its mouth.
	The tunnel: chevrons in the belt's color, pointing the way items travel.
	Input priority: a triangle at the input edge on the preferred side.
	Output priority: a triangle at the output edge on the preferred side.
	Filter: a diamond where the filtered item leaves; the rest leaves on the other side. One diamond color per item.
	Belt tier: a rim on splitters and a bar over a hood's mouth, in the tier's color (yellow, red, blue, turbo green).
	Side-loading: a belt that ends against the side of another.

Left and right are those of somebody travelling with the belt. The shaded lines of tiles are the entry and exit lines, when the layout has them. The glyphs above are `\beltglyph`, which sets one glyph in running text: `\beltglyph{entrance}` gives .

2 Front ends

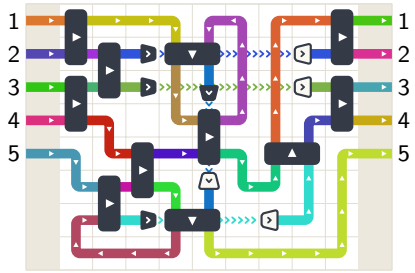
2.1 Files: `\beltlayout`

`\beltlayout[keys]{file}` reads, by the file's extension:

- `.json`: tiles in Factorio-SAT's format, the bare list `tiles[y][x]` or an object `{"tiles": ..., "gap": 4, "inputs": [...], "outputs": [...], "tier": ...}`;
- `.jsonl`: search records; the smallest record with status `sat`, or the one picked by `pick` (network file) and `record` (counted from 0);
- anything else: a blueprint string, a blueprint book, or the JSON inside one; `index` (from 0) or `label` (part of the label) picks a blueprint of a book.

Directions are numbered 8 ways in Factorio 1.1 and 16 ways in 2.0; the blueprint's `version` field decides, and `factorio=1.1` or `factorio=2.0` overrides it. A 2.0 splitter filter is an object with `name`, `quality` and `comparator`, and the game leaves out the last two when the filter takes any quality. It is read as its item name, with a note when the quality is not plain normal, as in *copper-ore (any quality)*, and written back as it was read.

```
\beltlayout[labels, flow=right]{examples/f04_11x8.txt}
```



2.2 Strings: `\beltblueprint`

A blueprint string pasted into the document. Line breaks and spaces inside it are ignored.

```
\beltblueprint[inputs={A,B}, outputs={left, right}]{0eNqtkMEKwjAQRH9F9rxKGxuh+RU
  RaXXVQJqEZCtK6b8bFUWsPYiewmZ2HrPTQW1a8kFbBtWBZmpAvfwhkGXNmiKoZXcfzmvbNjUFUDmCdzH
  ...}
```



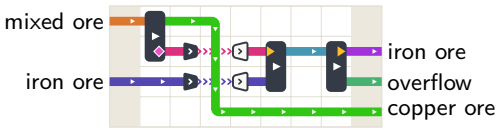
2.3 Typed grids: `beltgrid`

One cell per tile, cells separated by spaces, rows by `\\`. Keep a space between two `^` cells: TeX reads `^^` as a character code.

Cell	Tile	Cell	Tile
`> < ^ v`	belt	`S>`	a splitter tile, on both tiles
`I>`	underground entrance	`S>[in=left]`	with options, on either tile
`O>`	underground exit	`.` or `_`	empty

Splitter options are `in` and `out` (`left` or `right`) and `filter` (an item name); the filtered item leaves on the `out` side, left when `out` is not set. The tiles of a splitter facing right or left are stacked, those facing up or down sit side by side. A belt turns as in the game: when exactly one belt feeds it from a side and none from behind.

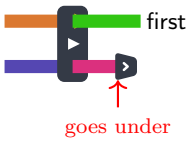
```
\begin{beltgrid}[inputs={mixed ore, iron ore},
    outputs={iron ore, overflow, copper ore}]
 >  S>[filter=iron-ore, out=right] > v .  .  ...\\
 .  S>  I> v O> S>[in=left] > S>[out=left] >  \\
 >  >   I> v O> S>          . S>           >  \\
 .  .   .  > >  >           > >            >
\end{beltgrid}
```



2.4 By hand: the TikZ library

`\usetikzlibrary{beltmap}` works without the package, in any engine. One unit is one tile and y points up; `(tile cs:x=3,y=5)` or `\bmtile{3}{5}` is the center of the tile in column 3, row 5, counted from the top left as in Factorio-SAT. The glyphs are pics whose argument names a direction and options: `bm head`, `bm entrance`, `bm exit`, and `bm splitter`, placed at the middle of its two tiles. Belts are paths with the style `bm belt=`*color*; `bmc0` to `bmc11` are the colors of the palette.

```
\begin{tikzpicture}[bm tile=6mm, bm theme=schematic]
  \path[bm belt=bmc0] (0,-0.5) -- (1.5,-0.5);
  \path[bm belt=bmc1] (0,-1.5) -- (1.5,-1.5);
  \pic at (1.5,-1) {bm splitter={right, out=left}};
  \path[bm belt=bmc2] (1.5,-0.5) -- (3,-0.5);
  \path[bm belt=bmc3] (1.5,-1.5) -- (2.5,-1.5);
  \pic at (2.5,-1.5) {bm entrance=right};
  \node[bm label, anchor=west] at (3,-0.5) {first};
  \draw[red, thick, <-] (2.5,-1.8) -- ++(0,-0.6)
    node[below, font=\scriptsize] {goes under};
\end{tikzpicture}
```

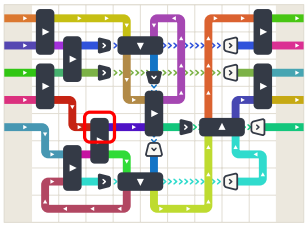


Pictures from the other front ends take TikZ code through the key `annotate`, drawn after the layout in the same coordinates:

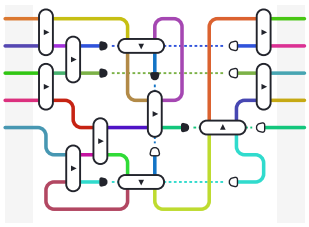
```
\beltlayout[annotate={\draw[red, very thick, rounded corners=2pt]
    (tile cs:x=1,y=4) ++(-0.55,0.55) rectangle ++(1.1,-1.1);}]{...}
```

3 Themes

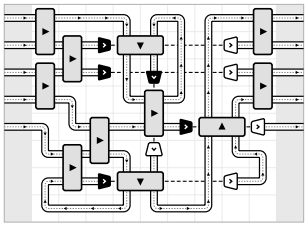
`theme` sets the look; the drawing does not change. The same layout, `f05_11x8.txt`, in each:



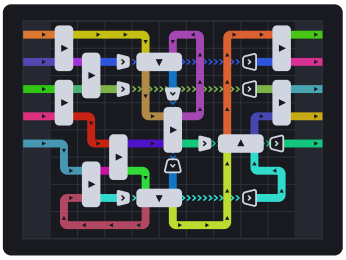
`schematic` (the default; with an `annotate` box)



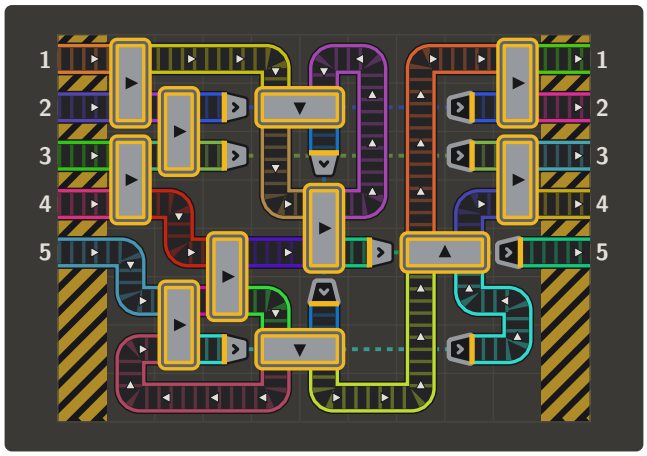
`transit`



`print`: black ink, both lanes of each belt



`dark`



`industrial`: after the feel of the game, drawn from scratch without its graphics

`industrial` draws each belt as a dark track with slats between two rails in the belt's color, splitters and underground belts as steel housings striped in their tier's color (yellow when the layout names none), and hazard stripes on the entry and exit lines.

4 Keys

Keys go in the optional argument of each command, in `\beltmapset{...}` for the rest of the group, or as package options.

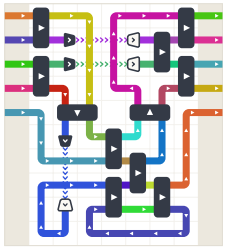
Key	Default	Effect
`theme`	`schematic`	`schematic`, `dark`, `transit`, `print`, or `industrial`
`tile`	`4.4mm`	size of one tile
`labels`	`false`	number the inputs and outputs
`inputs`, `outputs`		labels, in order from top to bottom or left to right; braces protect a comma
`flow`	as drawn	`right`, `up`, `left`, or `down`: turn the layout to run that way
`tier`		`yellow`, `red`, `blue`, or `turbo` for tiles that name none; turns the trim on
`trim`	`auto`	show tier colors on splitters and hoods, yellow for tiles without a tier; `auto` shows them when a tier is given or tiers are mixed, and always in `industrial`
`color`	`belt`	`belt`: one color per belt; `tier`: the color of its tier
`mix`	`none`	`blend` (alone, `mix` means this) or `bands`: color each belt by the inputs it carries, and judge the balance (section 8)
`mix lanes`	`false`	color the two lanes of each belt on their own (section 11)
`focus`		an input number: shade each belt by how much of that input it carries
`palette`	`wheel`	`wheel`, `okabe-ito`, `lightness`, or `gray` (the default in `print`)
`expect`		a list of `balanced`, `unbalanced`, `lanes`, `throughput`, `limited`: stop when the layout is not so (section 9)
`throughput`	`false`	run the max-flow test and mark the cut (section 10)
`view`	`layout`	`graph`: the splitter graph instead (section 12)
`numbers`	`false`	splitter numbers on the layout, as in the graph
`compare`		a layout file: mark the tiles that differ from it
`step`		one step of the flow, as in `\beltanimation`
`columns`, `limit`	3, all	cells per row and number of records of `\beltgallery`
`lanes`	`false`	a line between the two lanes of each belt (always on in `print`)
`grid`	`true`	the tile grid: `faint`, `true` (normal), `strong` (over the entry and exit lines too), or `false` (section 6)
`grid major`		a heavier line every n tiles, counted from the inside corner
`rulers`		number the tiles of the inside along the top and left; `rulers=5`: every fifth
`dimensions`	`false`	the size of the inside in tiles, below and to the left
`ends`, `heads`	`true`	the shaded entry and exit lines, the arrowheads
`gap`	file, else 10	longest tunnel for tiles without a tier
`pick`, `record`		which search record of a `.jsonl` file
`index`, `label`, `factorio`		which blueprint of a book; direction numbering, read and written
`blueprint`	`none`	`file`: write the picture's blueprint string beside the PDF; `attach` (alone, `blueprint` means this): also attach it to the PDF (section 14)
`blueprint label`	read, else *Belt layout*	the label of the written blueprint
`alt`	generated	alternative text in a tagged PDF
`annotate`		TikZ code drawn after the layout
`tikz`		options for the `tikzpicture`

5 Splitter options in four directions

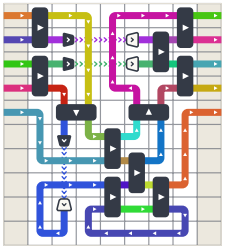
	in left	in right	out left	out right	filter left	filter right	in left, out right
right							
up							
left							
down							

6 Counting tiles

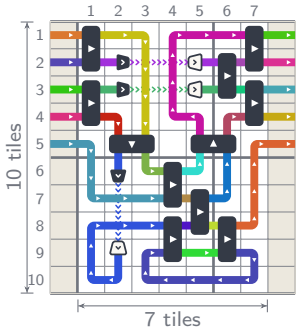
The size of a layout is counted inside its entry and exit lines, as the search records count it. `grid` sets how much the tile grid shows, from `faint` to `strong`, which also runs over the entry and exit lines. `grid major=5` draws a heavier line every five tiles from the inside corner, `rulers` numbers the inside tiles along the top and left (every one, or every fifth with `rulers=5`), and `dimensions` marks the inside size in tiles. `\beltmapcaption` gives the same size in words.



`grid=faint`

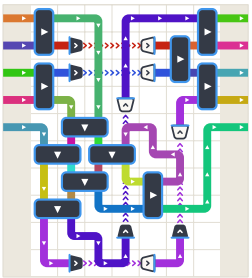


`grid=strong`

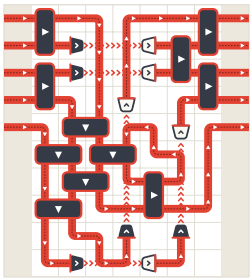


`grid=strong, grid major=5, rulers, dimensions`

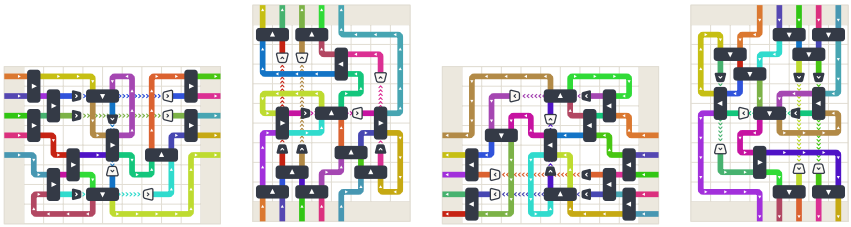
7 Tiers, lanes, and flow



`tier=blue`



`tier=red, color=tier, lanes`



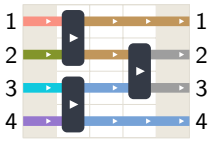
`flow=right` `up` `left` `down`

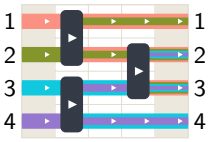
8 Mixing: is it a balancer?

With `mix`, each input carries an item of its own color, and every belt is colored by what it carries. The flow is computed exactly from the trace. Each input feeds one unit; a splitter sends half of what enters it out of each side, all of it to its output-priority side, or all of it to its one side that leads anywhere; a side-load adds to the belt it joins; a belt that leads nowhere carries nothing. A loop fed only through output-priority sides takes items in and lets none out; in the game it fills, and its splitter then sends everything to the other side. beltmap finds such loops, as sets of belts that no flow leaves, and counts them full. A layout is *balanced* when each of its m outputs carries $1/m$ of every input, and then all its outputs look alike. `\beltmapverdict` holds the verdict on the last picture, the alternative text includes it, and the log lists what each output carries.

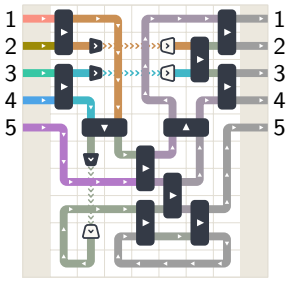
`mix=blend` paints each belt with the average of its input colors, each weighted by that input's share of the belt and taken in the OKLab color space. The input colors lie evenly round a circle of hues, so the even blend of all of them is gray, and a balancer ends in gray belts. The converse is false: gray outputs do not make a balancer. A color has three coordinates, lightness and two of hue, and a blend presses the n shares of a belt into these three, so from 5 inputs up many uneven mixes blend to exactly the gray of the even one. With 5 inputs, a belt that carries 12.2%, 15.5%, 35.2%, 0%, and 37.2% of inputs 1 to 5 is that gray. Up to 4 inputs different mixes blend to different colors, but close mixes give colors the eye cannot tell apart. A tinted output does carry an uneven mix; which inputs it carries too much of is read from the bands or from `\beltmapmatrix`.

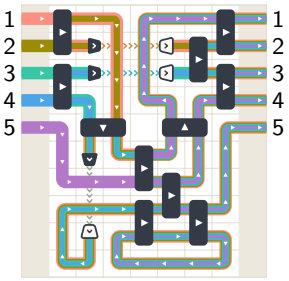
`mix=bands` draws each belt as stripes, a pair per input, each pair as wide as that input's share of the belt. Bands tell any two mixes apart, but they show shares, not amounts: a belt that carries half a unit and one that carries two units in the same proportions get the same stripes. The outputs of a balancer carry the same stripes in the same widths, and so do the outputs of a layout that sends the same mix to its outputs in unequal amounts, as in the last picture below. Neither view decides balance. The verdict does: `\beltmapverdict` is computed from the amounts each output carries, and `expect=balanced` stops the document when the layout is not balanced (section 9).



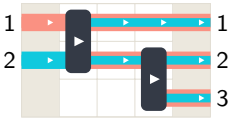


`mix` and `mix=bands`: not balanced: output 1 carries 50% of input 1, not 25%





balanced: each of the 5 outputs carries 20% of each of the 5 inputs

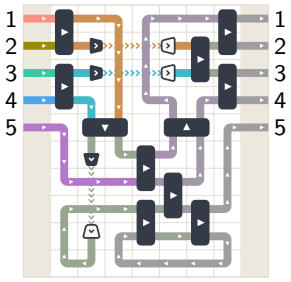


The same stripes on every output, but output 1 carries twice as much as each of the others.
not balanced: output 1 carries 50% of input 1, not 33.3%

9 Exact shares, claims, and captions

The shares are computed in exact fractions, and in floating point only when the fractions outgrow 62 bits. After each picture, `\beltmapmatrix` sets them as a table, a row per input and a column per output. `expect` turns a claim of the text into a check: the document stops with an error when the layout is not what it says. It takes `balanced`, `unbalanced`, `lanes` (section 11), `throughput`, or `limited` (section 10), or a list of them. `expect` works without `mix`: it computes what it checks.

`\beltmapcaption` is a one-line summary of the last picture: inputs and outputs, the size between the entry and exit lines, splitters, tunnels, and every verdict the picture computed. It is the same size the search records give as their area.

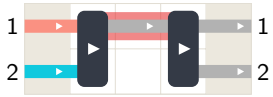


	output				
input	1	2	3	4	5
1	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
2	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
3	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
4	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
5	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$

5–5 · 7×10 · 11 splitters · 3 tunnels · balanced · passes max-flow

10 Throughput

A balancer can balance and still not pass full belts: a belt that every input must cross caps them all. `throughput` runs the max-flow test. Every lane is given a capacity of one, every choice of open inputs I and outputs O is tried, and a choice is *limited* when less than $\min(|I|, |O|)$ belts can get through. A limited choice is proved limited: the test finds a *cut*, a set of lanes whose removal separates I from O, with less capacity than the choice needs. The picture shows the cut of the worst choice glowing under its belts, and `\beltmapthroughput` holds the verdict. Passing the test is necessary for a throughput-unlimited balancer, not sufficient: a splitter does not route along a maximum flow, and the steady states that decide it (`design-methods/tu.py` of this project) need a mixed-integer program that LuaTeX does not have. Up to 4096 choices are tried; beyond that, only the choices with as many outputs as inputs.

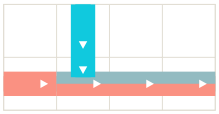


balanced, but limited: inputs 1, 2 to outputs 1, 2 get at most 1 of 2 belts (the 2 lanes marked form a cut)

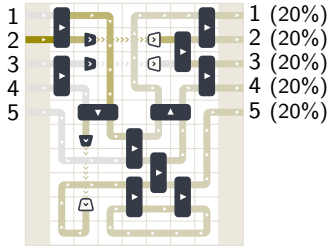
11 Lanes, focus, and palettes

The shares are traced lane by lane: belts, turns, underground belts, and splitters keep each lane, and a side-load puts both lanes of the incoming belt onto the near lane of the belt it joins. `mix lanes` colors the two lanes of each belt on their own, and `\beltmaplanes` says whether the lanes are kept: whether each output lane carries $1/m$ of each input lane on its own side and nothing of the other. `expect=lanes` checks it.

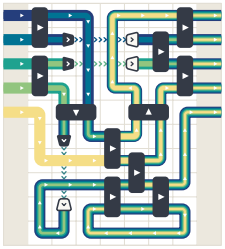
`focus=`i follows one input: each belt is shaded by how much of it is input i, and the output labels give the share of input i that reaches each output. `palette` picks the input colors. `wheel`, the default, puts the hues evenly round a circle, so the even blend is gray. `okabe-ito` uses Okabe and Ito's colors for readers with color blindness. `lightness` runs from dark blue to light yellow, told apart by lightness alone. `gray` runs from black to light gray, and is the default in `print`. Away from `wheel`, a balancer's outputs still look alike, but their color is not gray.



`mix lanes`: a side-load



`focus=2`

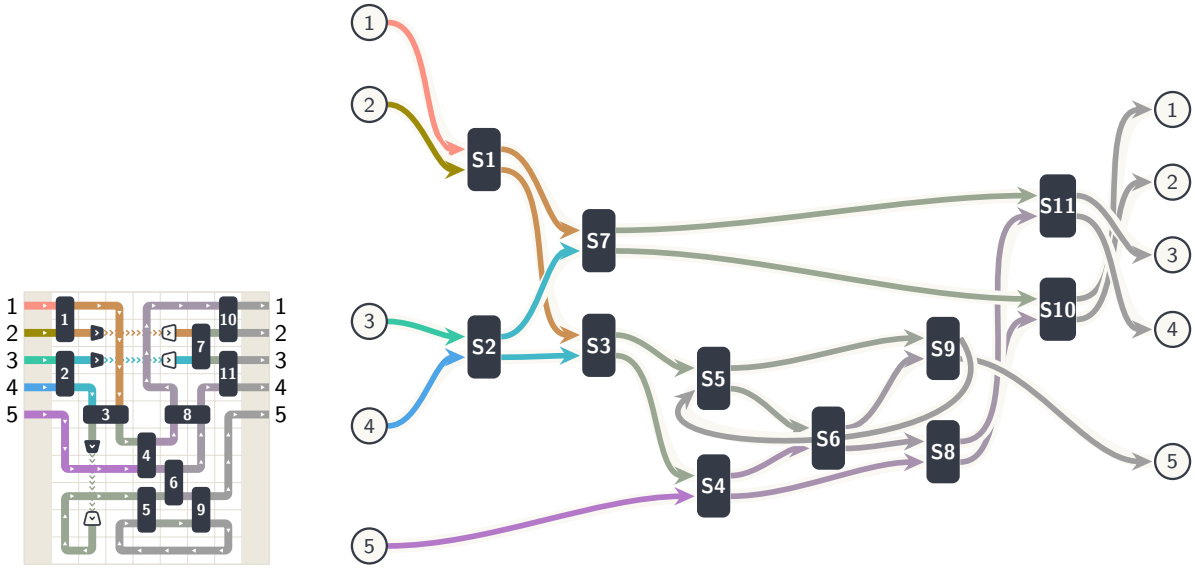


`palette=lightness`

12 The splitter graph

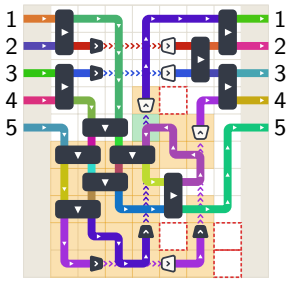
`view=graph` draws the network instead of the layout, with none of the layout's positions. Each input, splitter, and output is a node, and so is every point where a belt side-loads. Each piece of belt is an arrow, in the color the layout gives it, `mix` included. The inputs stand in a column on the left and the

outputs in one on the right. Each splitter takes the column of its longest path from the inputs, once the loops are cut. A force layout then sets the heights: every pair of nodes repels, every belt pulls its ends together, and the inputs and outputs fall in whatever order this gives. Arrows leave a node to the right and enter the next from the left, each at a point of its own, and an arrow that closes a loop swings round below or above. `numbers` writes each splitter's number on the layout in place of its arrow, the number its node carries in the graph.



13 Comparing two layouts

`compare=`*file* draws the layout and marks how it differs from the layout in the file. Tiles the other layout lacks are shaded green, tiles that hold something else are shaded amber, and tiles only the other layout uses are outlined in red. The two are lined up by their top left entities; `flow` turns both.



f01 compared with f00: 5–5 · 7×10 · 11 splitters · 5 tunnels · +1 −4 ∼29 tiles

14 Blueprints out

Every picture also has a blueprint: after each one, `\beltmapstring` holds the blueprint string of the layout as drawn (turned by `flow`, tiers given by `tier`), and `\beltmapprintstring[width]` sets it in small type that may break after any character. The string uses Factorio 2.0's numbering unless `factorio=1.1` is given or the layout was read from a 1.1 string; tiles without a tier become plain (yellow) entities. With `blueprint=file` the string is also written to `\jobname-blueprint-`*n*`.txt`, and

with `blueprint` it is moreover attached to the PDF, where a viewer lists it among the attachments. Copying the attachment, or the file, is the safe way into the game: text copied from the page may carry line breaks.

A blueprint stores no belt turns: the game works them out from the neighbors. Each string is read back as soon as it is written, and a layout whose turns would come out differently in the game draws a warning. On every example the string reads back to the same layout, and for the examples that are blueprint strings it holds the same entities as the string it came from.

The strings were tested in Factorio 2.1.24 (with Space Age), headless. A small mod imported each string, built it, read every entity back through the game's API (name, tile, direction, belt shape, underground type and partner, splitter priorities and filter), and exported a blueprint of what it built; beltmap's reading of the string was compared with the entities, and its reading of the export with the entities. All 53 cases passed, in 2.0 and 1.1 numbering, among them grids with every tier, every direction, turns, underground belts at full reach, priorities, and filters.




```
0eNqVUtFqwzAM/JWhZ280bl2IH/cbY5Sk1TqBIxtbGQOh/z412RiMQpsng+50d5I8Qht6
TJlYwI+ALCSEBfzbCCfKeBSKDH5nFmg4cN+1mMFXBrjpEDxIbrikmOW5xSBgIMVCS9sIF
/CbF2dgmN9pMndk7cOydo3sdnXa6hFZrRCnXg66wJgVUYOAH1fZH7uSAoko1ODs5R810/
nzRoTqN8L9AG71uua53jX3MfJyZuITXuaDFjpzE64Nt0WXRsFOob9vYyAOCmvtVUlPoRl
0Uq1+YS6zv9vbelfXzlm339rNNH0Di1LRXzEu
```

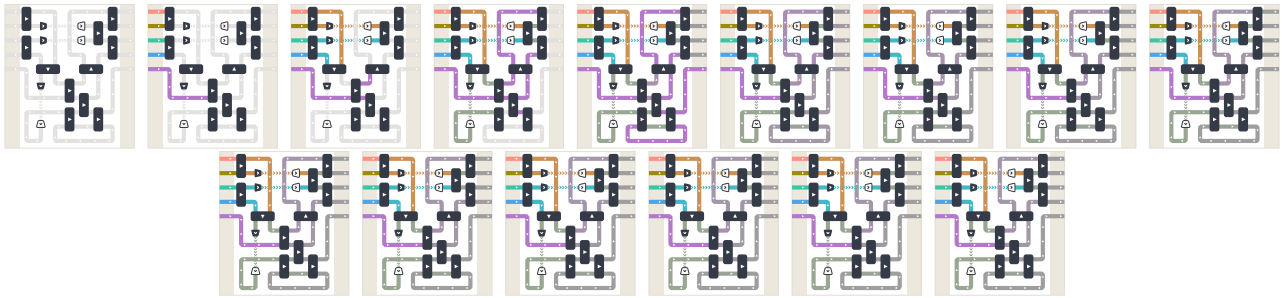

15 Books, galleries, and pictures outside LaTeX

`\beltmapbook[label]` collects every blueprint written with `blueprint=file` or `blueprint` into one blueprint book. At the end of the document the book goes to `\jobname-book.txt` and, attached to the PDF, into the viewer's attachments. One paste brings every layout of the document into the game.

`\beltgallery[keys]{file.jsonl}` draws every record with status `sat` of a search file: `columns` cells to a row (3 by default), each with its network file, size and area, and `\beltmapcaption`. `pick` keeps one network file and `limit` the first so many records; the other keys apply to every cell.

`\beltanimation[keys]{file}` draws one picture per step of the flow. The first has empty belts; each step moves the inputs' colors one piece of belt on, and the last is the steady state. `step=`k draws a single step. The script `beltmap_export.py` makes files for posting outside LaTeX: a PNG (`--png`, at `--dpi`), an SVG with text as paths (`--svg`), or an animated GIF of the steps (`--gif`). For example:

```
python beltmap_export.py examples/f00_9x10.json --keys "mix, labels" --gif f00.gif
```



16 Accessibility

In a tagged document (`\DocumentMetadata{tagging=on, ...}`) every picture is a figure with alternative text. Without `alt`, the text is generated from the trace, for instance: *Belt layout of 9 by 10 tiles with 11 splitters and 3 underground tunnels; 5 inputs enter on the left and 5 outputs leave on the right.* With `mix`, `mix lanes`, `throughput`, or `compare` it also gives their verdicts; `alt` replaces it. This manual is built tagged and passes veraPDF's PDF/UA-2 profile.

17 Installing

Put `beltmap.sty`, `beltmap.lua`, `beltmap-net.lua`, and `tikzlibrarybeltmap.code.tex` where LuaLaTeX finds them: beside the document, or in `tex/latex/beltmap` of a local `texmf` tree.

18 Known limitations

- LuaLaTeX only: the package stops with a message under pdfLaTeX or XeLaTeX. The TikZ library alone runs anywhere.
- A side-load onto an underground belt is traced as one onto a belt, onto the near lane; the game's rule there was not checked. No layout with a side-load was measured in the game.
- Blueprints were tested in Factorio 2.1.24 only. Strings in 1.1 numbering went through that version's import, not into Factorio 1.1, and no 2.0 release was tried. Written in 1.1 numbering, a filter loses its quality, which 1.1 does not have.
- Entities other than belts, underground belts, and splitters are left out, and counted in the log.
- Labels of a layout that runs up or down overlap when longer than a tile is wide.
- In `transit`, belts carry no arrowheads; direction shows on splitters and portals only.
- `mix` has no belt capacity: it judges whether a layout balances, and `throughput` only runs the max-flow test, which is necessary for full throughput but not sufficient. Filters are ignored: every item is the same item.