

Play Ball! with SVG and XSLT

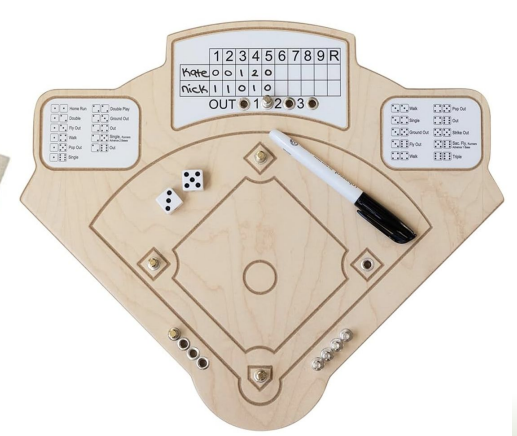
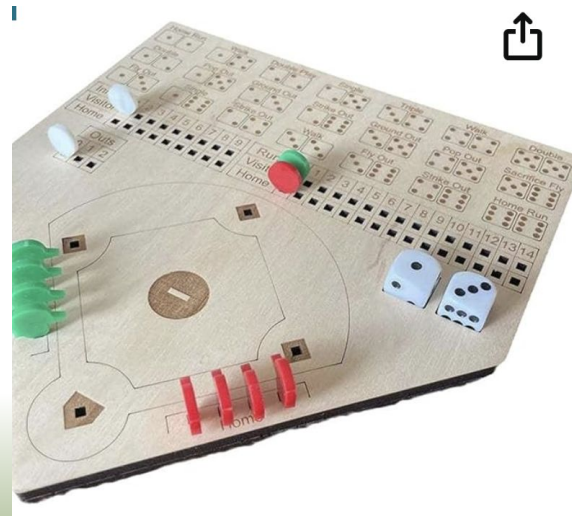
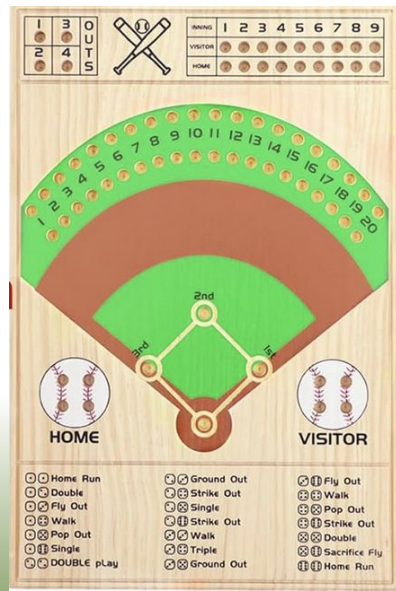
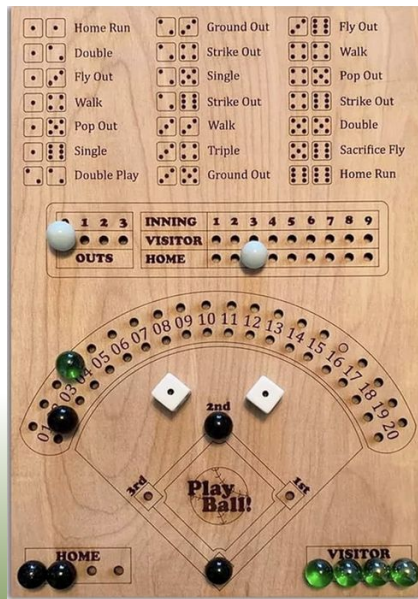
G. Ken Holman



Background

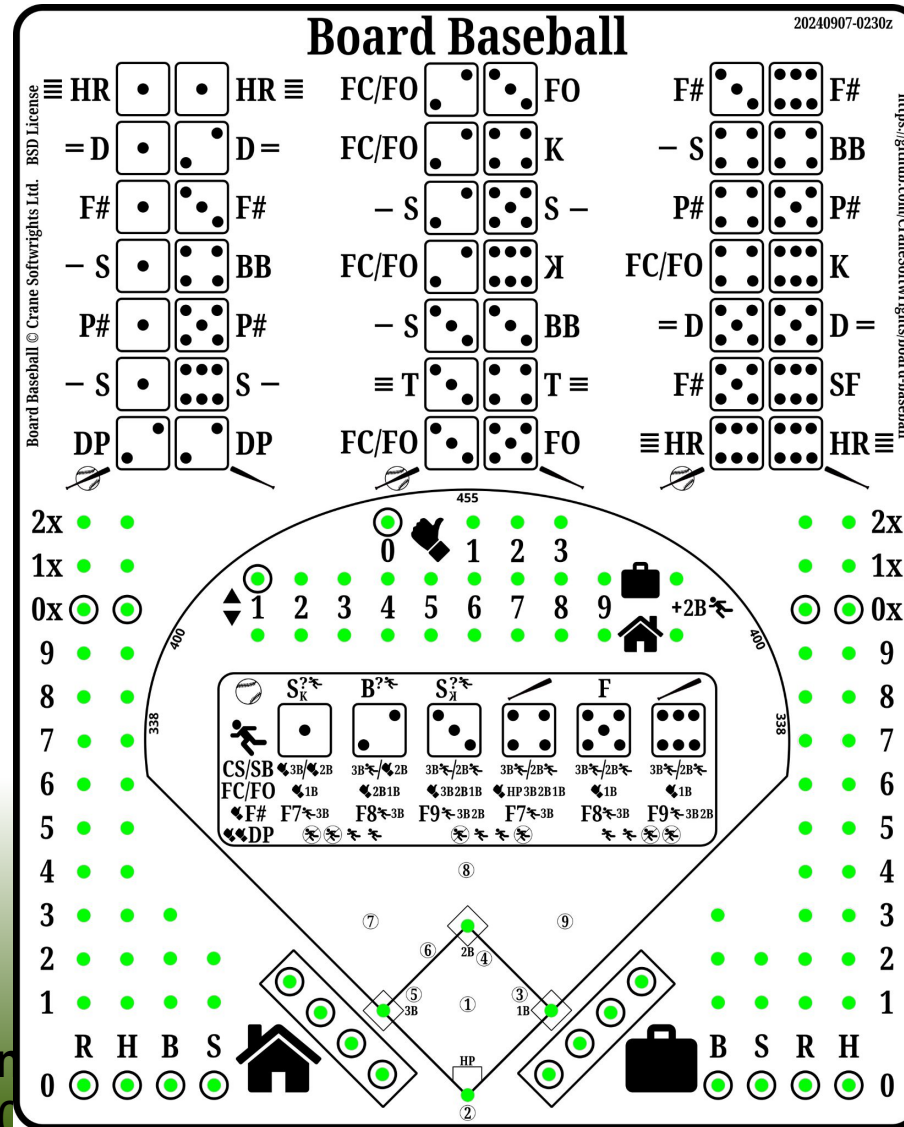
Asked by my son to teach him and my grandson baseball

- available games are unsuitable
- no pitching, few running distinctions
- not up to date with latest MLB major rule changes
- no MLB notation that is used in summary box scores



Background (cont.)

- I used Inkscape to create SVG for a playing surface on paper
- dress pins go through the paper to a foam backing



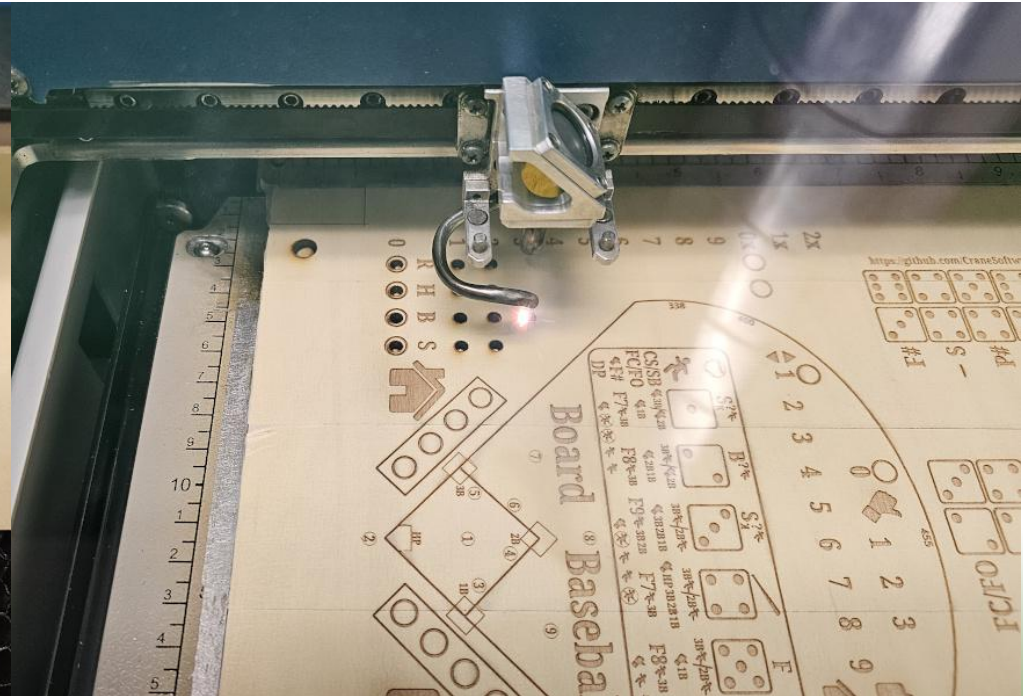
Background (cont.)

Free use of a laser cutter at the Ottawa Public Library

- cuts and burns wood and acrylic materials
- input is a single SVG file interpreted for the laser
 - .001in lines cut through the material (up to 1/4" thick)
 - all other non-white content is shallow-burned as a raster image



[YouTube snippet](#)



[YouTube snippet](#)

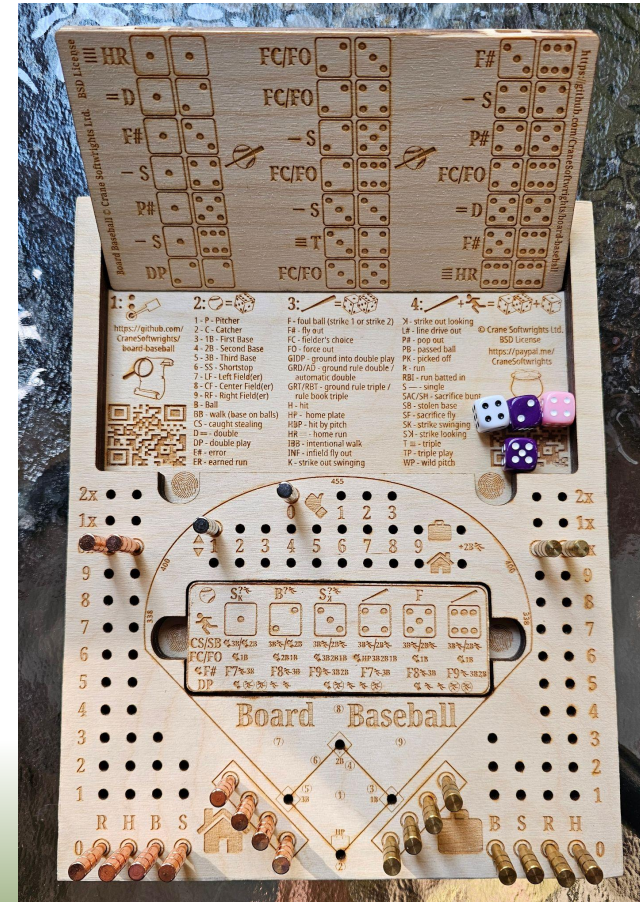
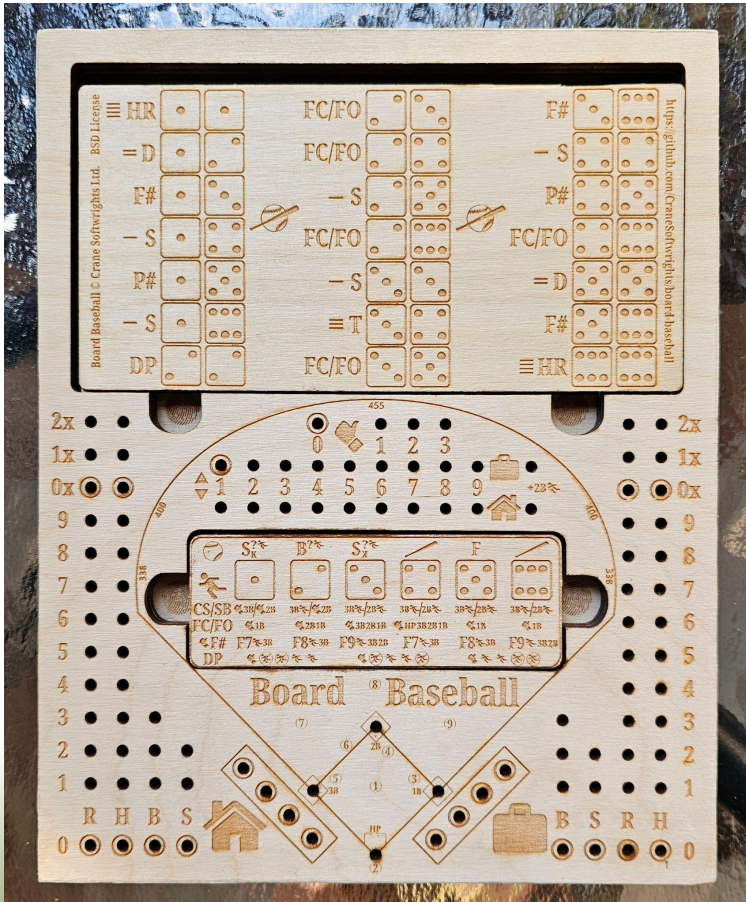
Play Ball! ... with SVG and XSLT

Declarative Amsterdam 2024-11-07 (2024-10-16 15:50Z)



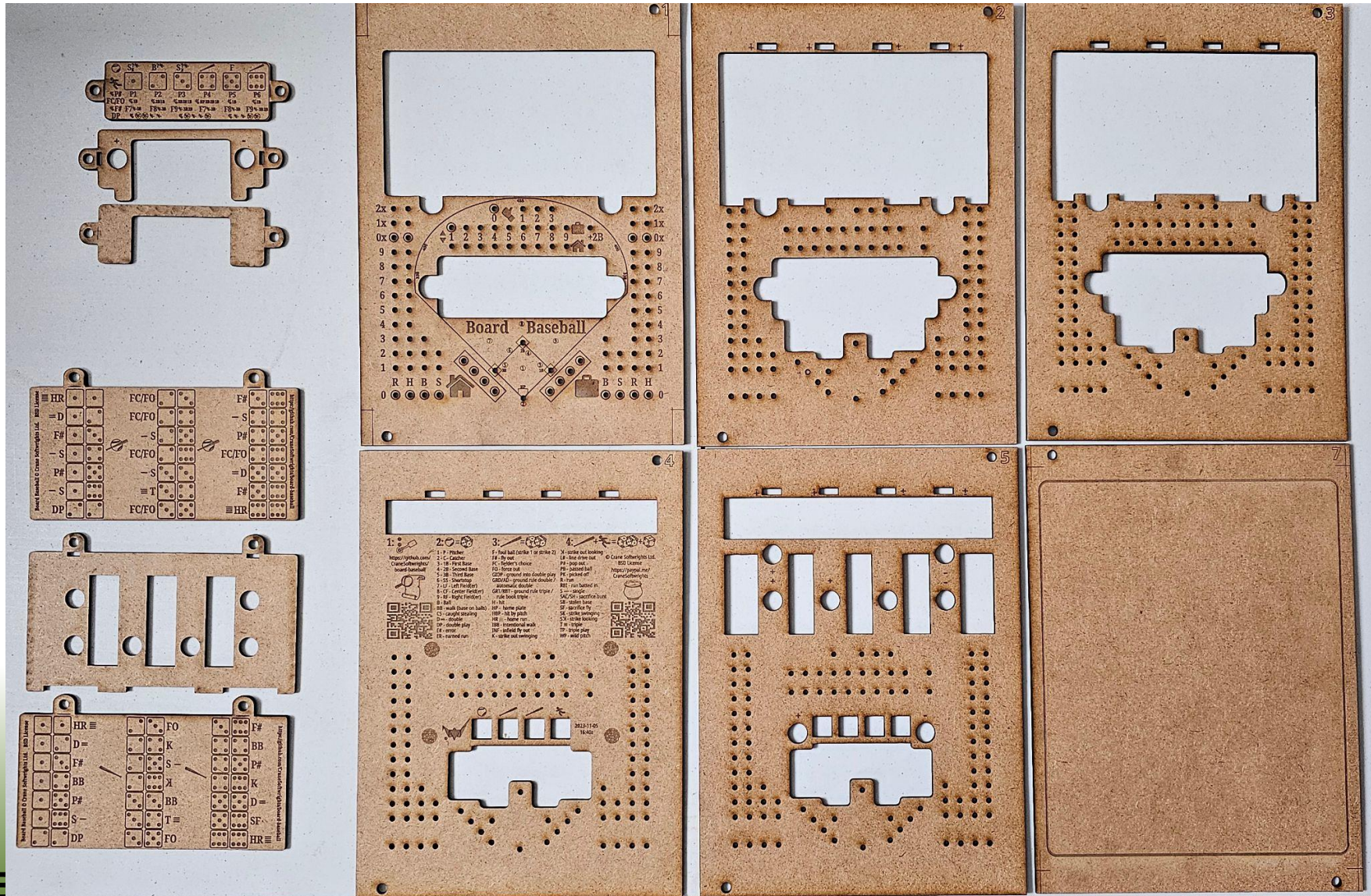
The challenge

A three-dimensional board has storage and play options:
- earth magnets help parts stay together when closed



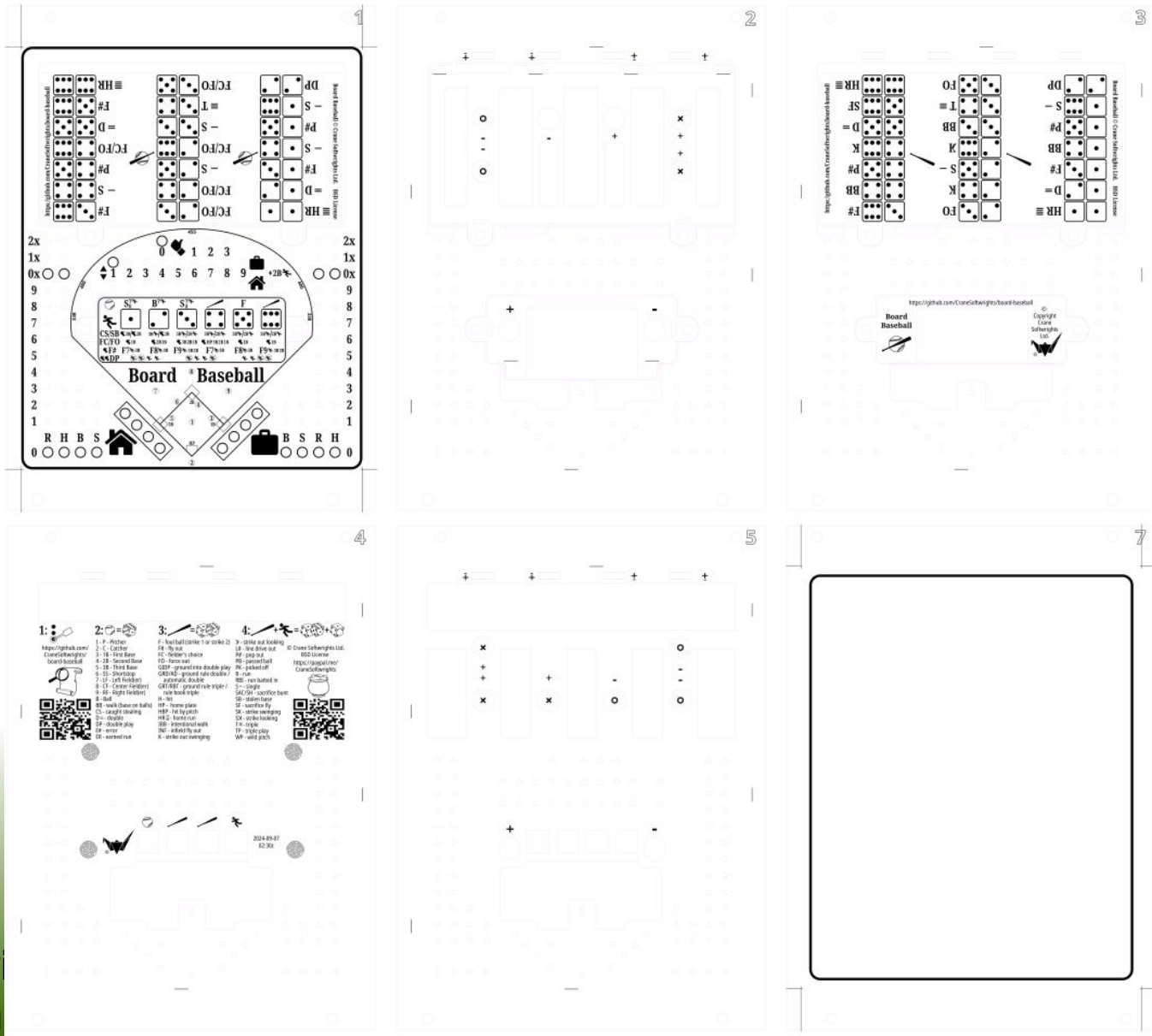
The challenge (cont.)

How to precisely coordinate creating six different boards?



The challenge (cont.)

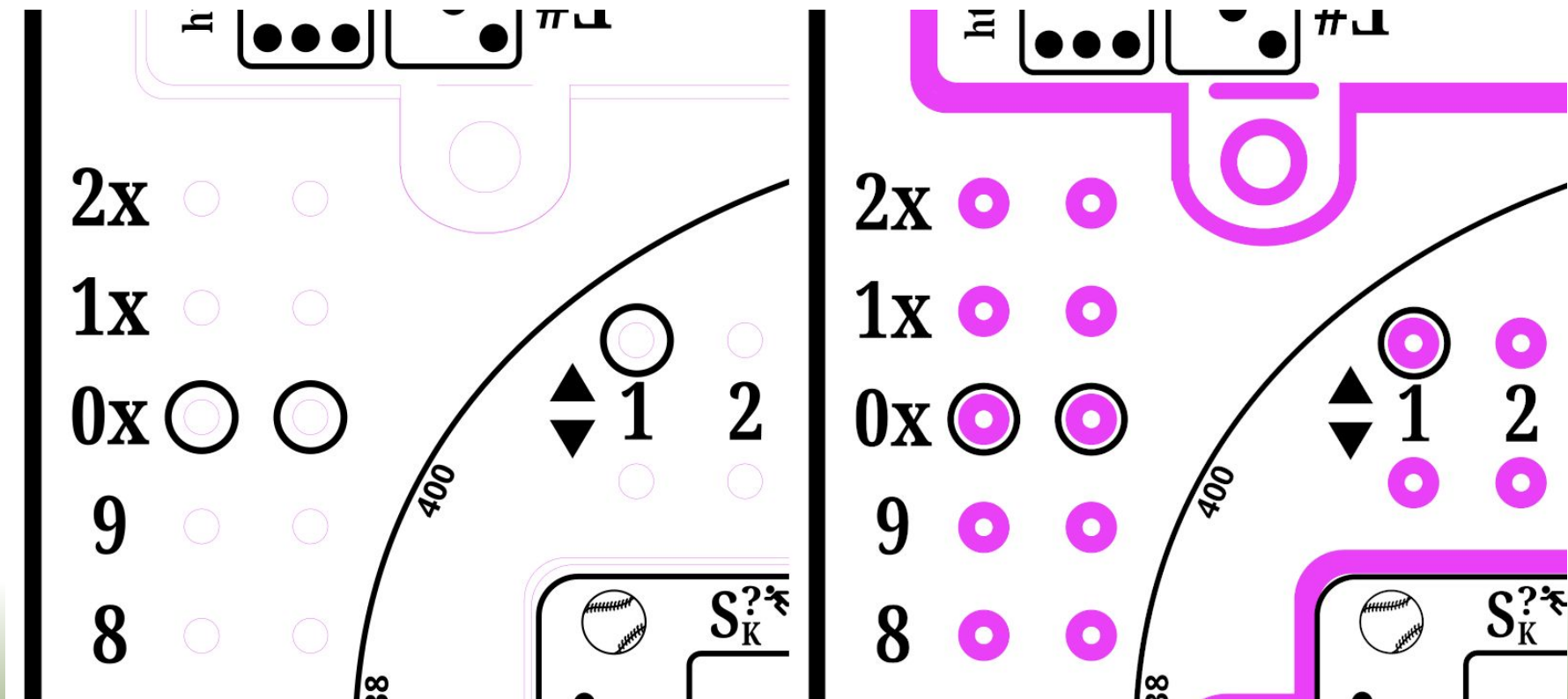
How to precisely coordinate creating six SVG files?



The challenge (cont.)

The .001in lines demand special attention during design

- need to distinguish from raster (I use magenta colour)
- the physical effect of the cut is close to 1mm



Devising and imposing a declarative scheme

Determine a successful system for creating desired results

- mimic a successful (labourious!) manual process to be automated
 - make each level's layers visible
 - write out the level's preview files
 - modify stroke widths as required by the laser cutter
 - write out the level's laser cutter files

Semantic operators and operands for the repeatable processes

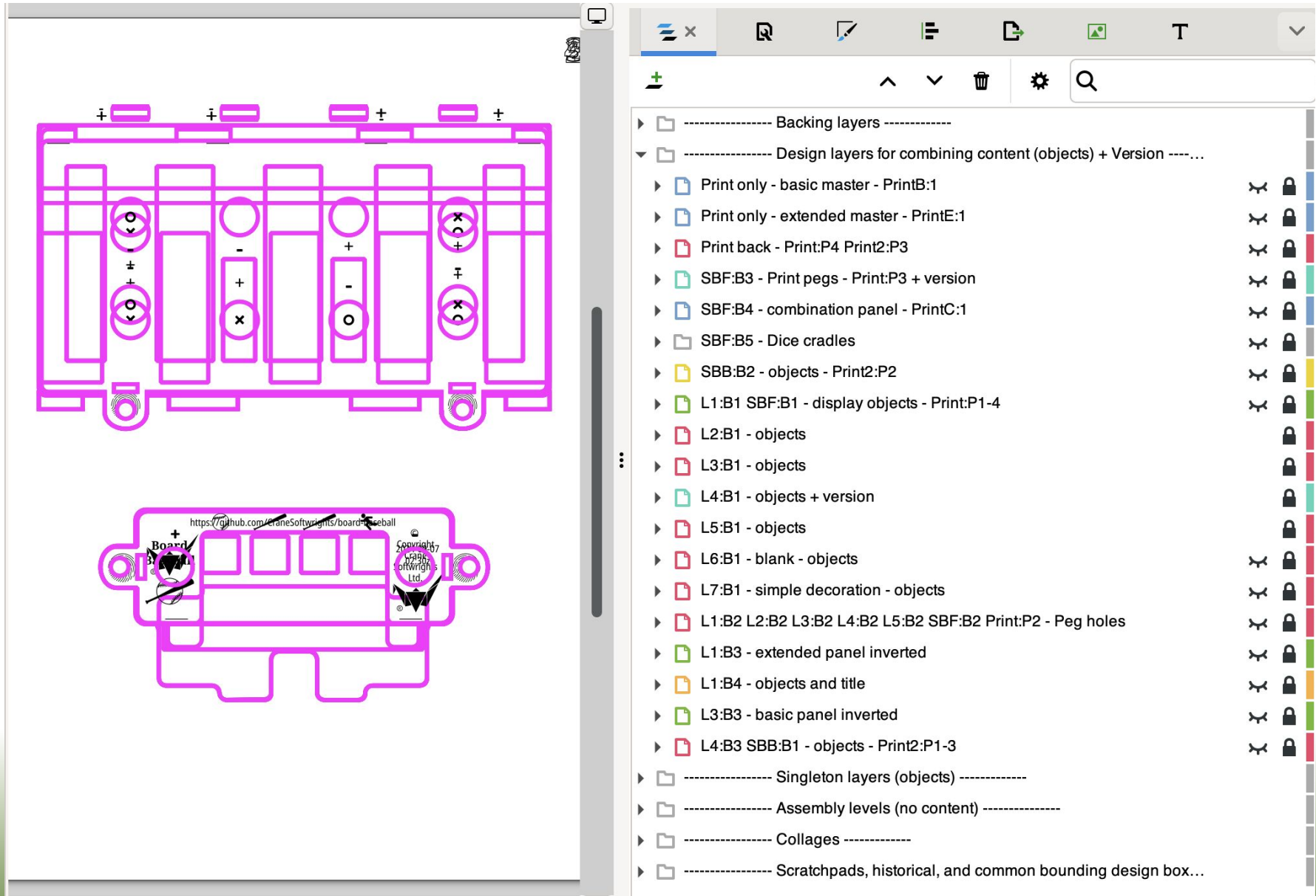
- identify layers and groups to be used in levels
- assemble multiple layers into a level combination assembly
- perform post-assemble tasks on the resulting level assemblies
- manage the output process to organize the results in different subdirectories

Leverage Inkscape layer/group labels for the declarative syntax

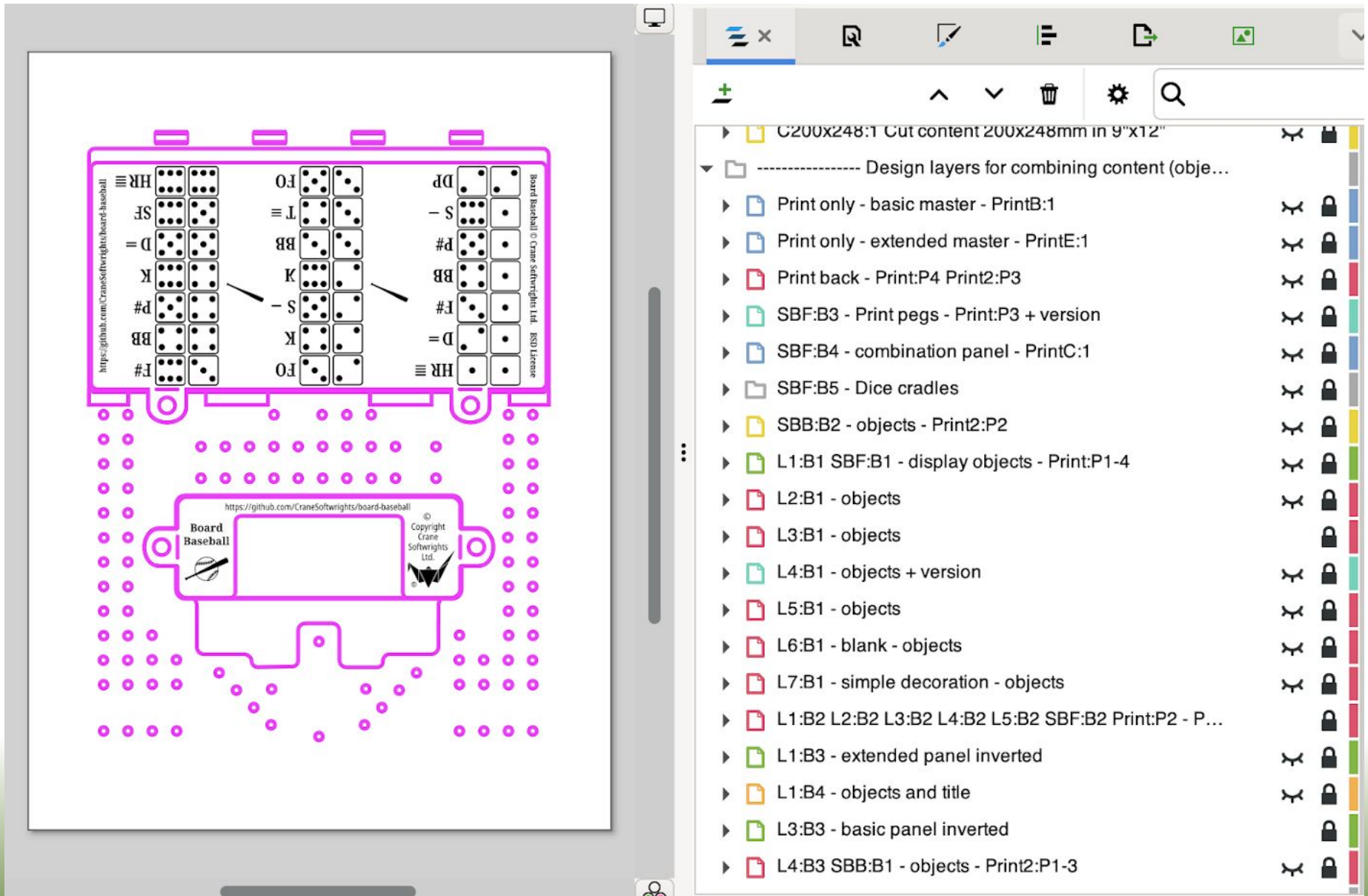
- all tokens containing ":" imply the layer is used in an assembly
- second token including "=" implies an assemble directive
 - augment "=" with post-assemble directives
- first token ending in "/" names a new subdirectory
 - the remainder of the tokens are put into a README.txt file



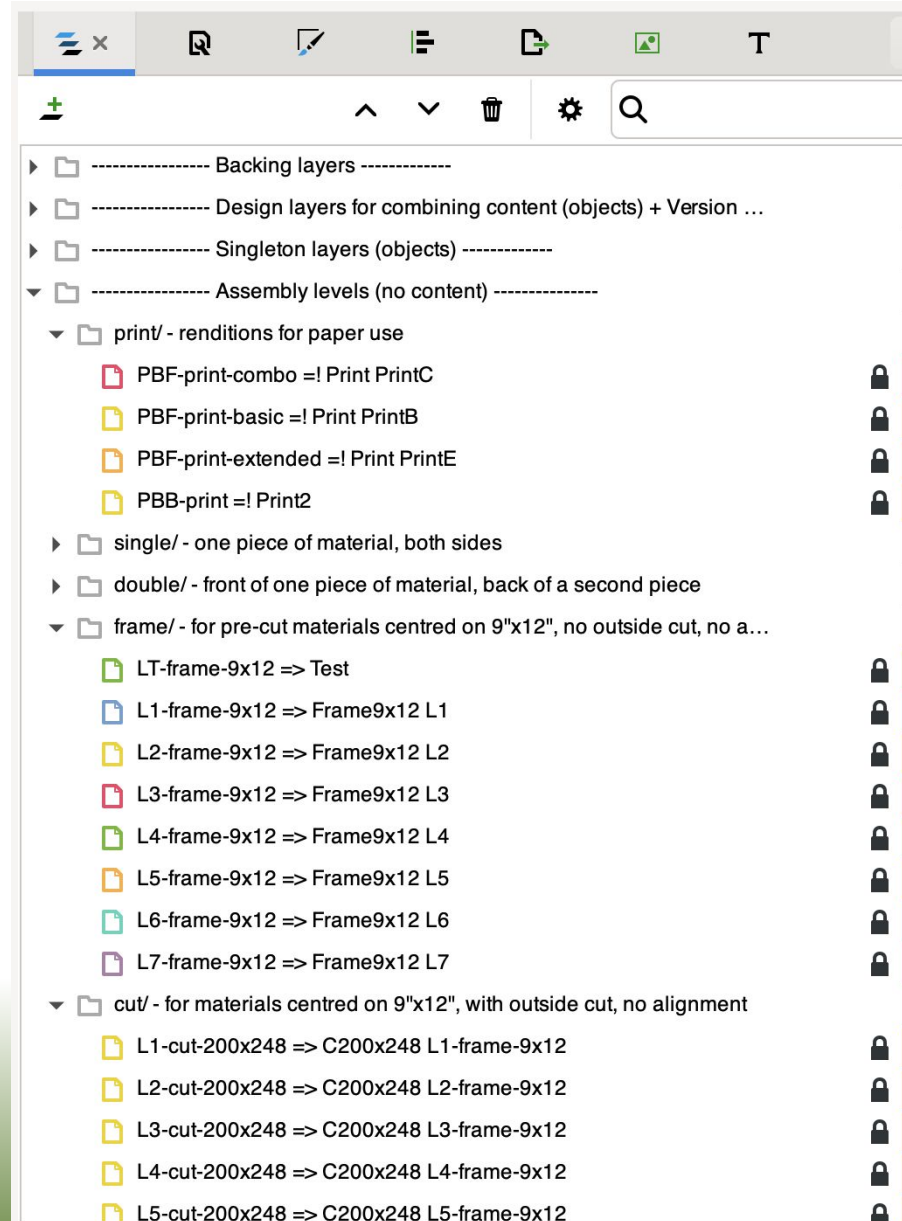
Selectively viewing multiple layers



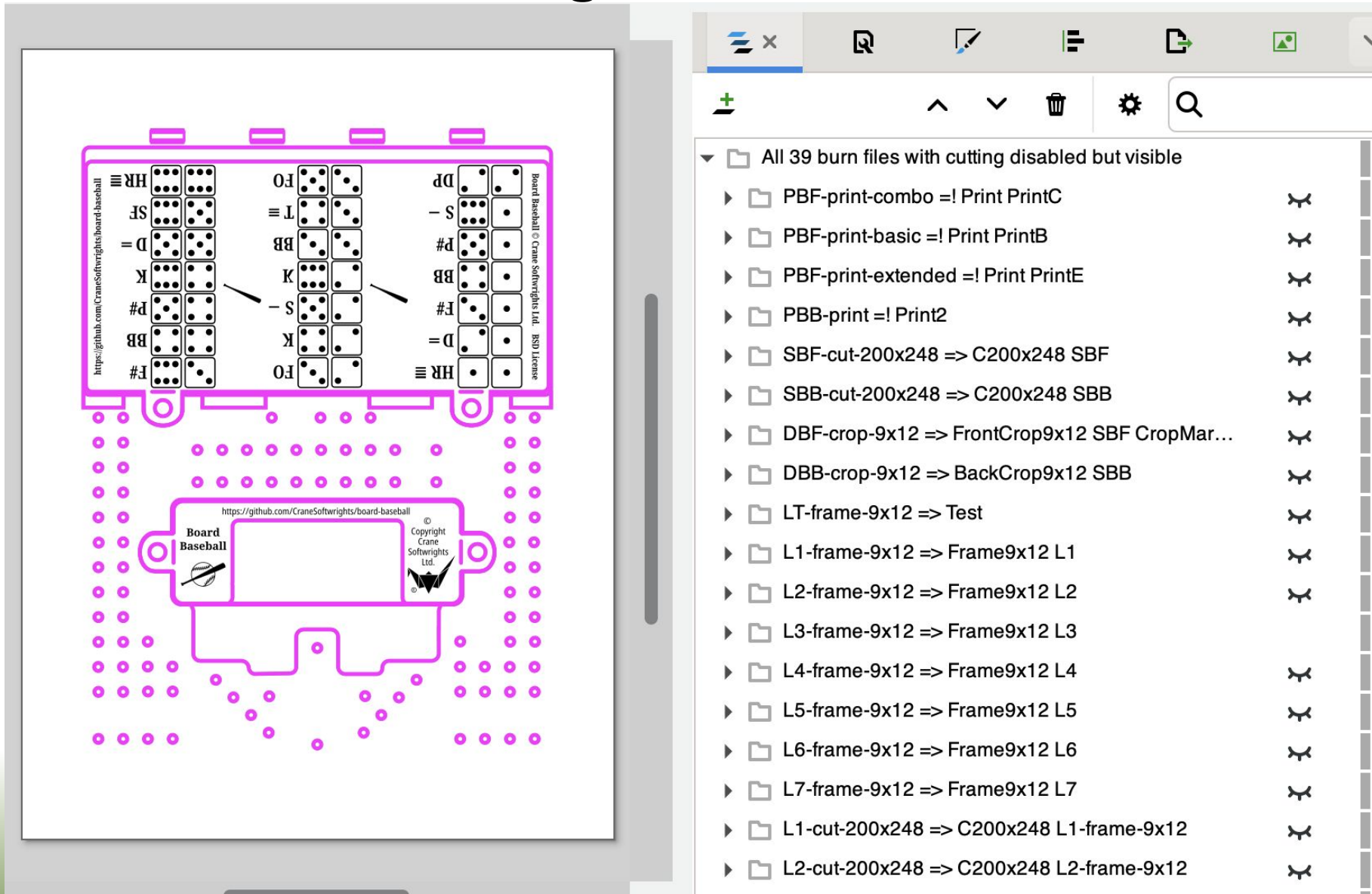
Manually viewing multiple layers for a level



Declarative syntax for semantics and operands



XSLT creates a single review SVG of all levels

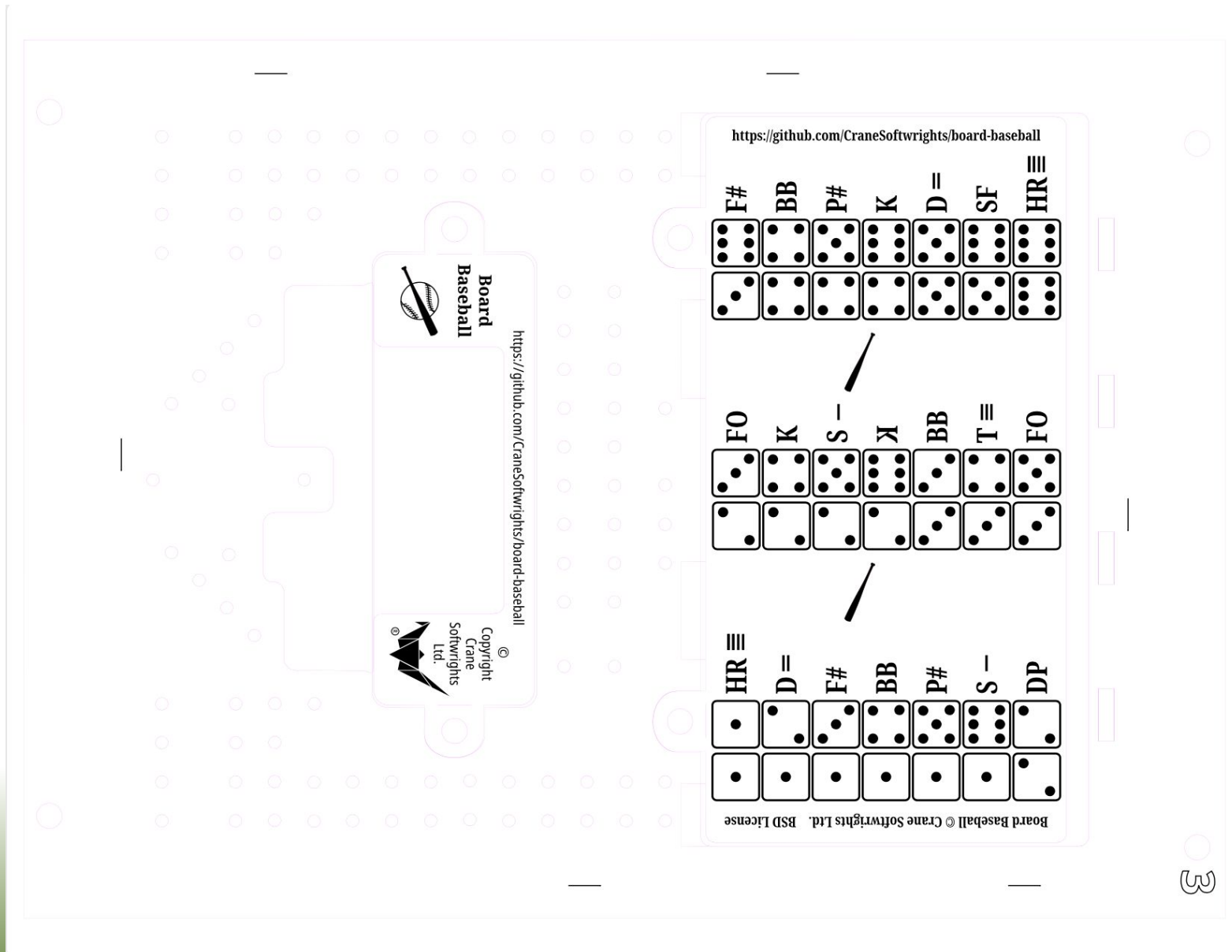


The image displays a software interface for PCB layout review. On the left, a large window shows a top-down view of a PCB layout. The layout is outlined in purple and features a central rectangular area with a grid of circular holes. Above this area, there are several rows of components, each represented by a small grid of dots. The components are labeled with alphanumeric codes: HR, SF, D, K, #P, BB, #F, and FO. The layout is surrounded by a grid of small circles, likely representing a drill or hole pattern. On the right, a smaller window shows a file list with a search bar and various icons. The file list is titled "All 39 burn files with cutting disabled but visible" and contains the following items:

- ▶ PBF-print-combo ⇒ Print PrintC
- ▶ PBF-print-basic ⇒ Print PrintB
- ▶ PBF-print-extended ⇒ Print PrintE
- ▶ PBB-print ⇒ Print2
- ▶ SBF-cut-200x248 ⇒ C200x248 SBF
- ▶ SBB-cut-200x248 ⇒ C200x248 SBB
- ▶ DBF-crop-9x12 ⇒ FrontCrop9x12 SBF CropMar...
- ▶ DBB-crop-9x12 ⇒ BackCrop9x12 SBB
- ▶ LT-frame-9x12 ⇒ Test
- ▶ L1-frame-9x12 ⇒ Frame9x12 L1
- ▶ L2-frame-9x12 ⇒ Frame9x12 L2
- ▶ L3-frame-9x12 ⇒ Frame9x12 L3
- ▶ L4-frame-9x12 ⇒ Frame9x12 L4
- ▶ L5-frame-9x12 ⇒ Frame9x12 L5
- ▶ L6-frame-9x12 ⇒ Frame9x12 L6
- ▶ L7-frame-9x12 ⇒ Frame9x12 L7
- ▶ L1-cut-200x248 ⇒ C200x248 L1-frame-9x12
- ▶ L2-cut-200x248 ⇒ C200x248 L2-frame-9x12



XSLT creates individual burn SVG for each level



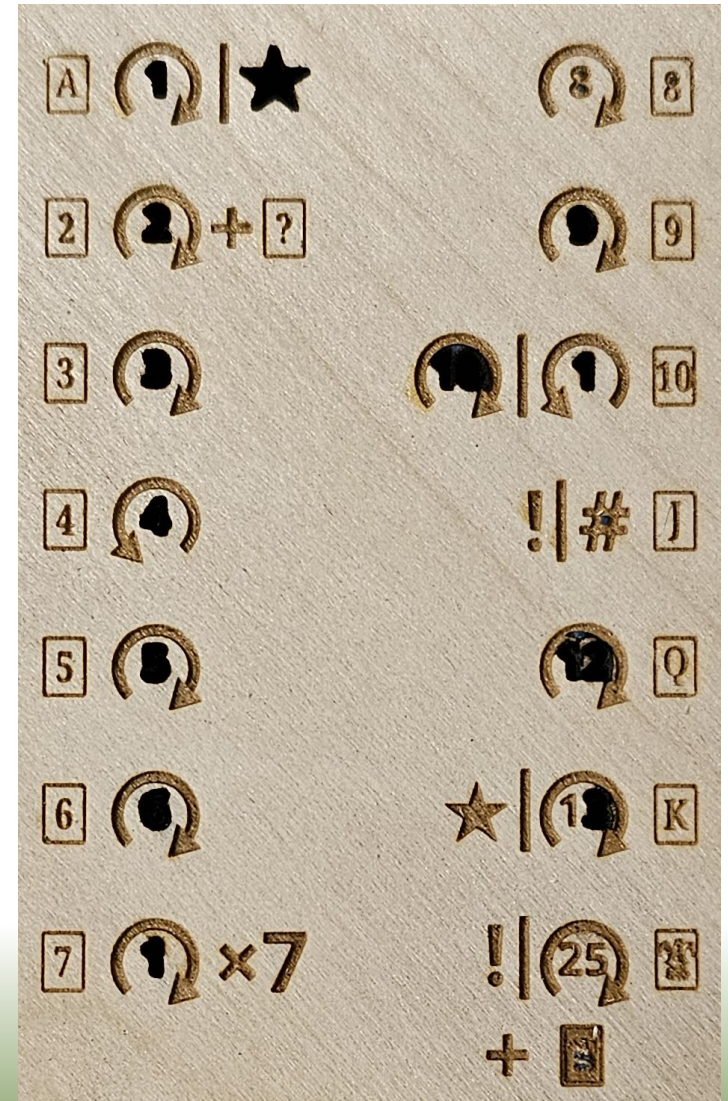
Leveraging re-use to create playing cards



Two XSLT tools used in the process

convertBadStrokes4designSVG.xsl

- any non-magenta stroke less than .001in is made thicker
- protects result from free SVG library images with strokes around .001in that end up cutting the final work instead of rasterizing
- protects transformed shapes from ending up with strokes that are .001in
- saves money by preventing wasted materials!



Two XSLT tools used in the process

`designSVG2burnFiles.xsl`

- **perform the heavy lifting processing a design SVG file:**
 - **interpret the declarative syntax to assemble the levels from the layers**
 - **create the single review SVG containing all of the resulting assemblies, but still in "design mode" for visual interpretation**
 - **create all of the SVG files of the resulting assemblies in "burn mode" for laser cutter interpretation**
 - **convert all magenta strokes to .001in width**
 - **create a shell script to invoke Inkscape to create each burn file PDF and PNG from the SVG file**



Leveraging GitHub actions

Every push to the game repository performs the following:

- install the required fonts - Noto Sans, Noto Serif
- install the required tools - Inkscape, SaxonHE
- install the designSVG2burnFiles.xsl stylesheet and its dependencies from Crane's git repository
- build all of the resulting print and burn files invoking the synthesized script
- update the repository with the latest PNG review files
- package the results for download from the actions menu
- create a release package and post it for download (for the shared develop, qa, and main branches, not for other user-specific branches)



Declarative SVG + XSLT = flexibility and control

Imposing a declarative scheme allowed me to:

- leverage the available title construct as data-driven
- streamline the simultaneous development of multiple "final" SVG images using a single "design" SVG image saving tremendous amounts of time and effort
- promote consistency and re-use during design for interoperable physical results in real-world materials

Using XSLT on the XML of SVG allowed me to:

- automate finicky steps to produce complex results
- save money by limiting failed burns



