Language Reference for FLIPER (FLIPping book compilER)

Samvid Mistry

November 28, 2020

Contents

1 Grammar

The language comes with the following primitives for easy creation of flipbooks.

- `CNV` *width height* [*red green blue alpha*]
  Canvas: This command is used to set the width and height for the canvas that will be used to draw the flipbook. Optionally, you can provide a color with `RGBA` values, which will be used as background color. If color components are not specified, the created canvas will have a white background.

- `IMG` *path id* [*six siy*]
  Image: This command is used to load an image present at path specified by parameter *path* into the identifier/register specified by *id*. *path* and *id* both are strings and all strings are enclosed in double quotes(") in FLIPER. Both absolute as well as relative paths should work as value for *path*, however, FLIPER has been tested mainly with images existing in the current directory. Optionally, you can provide the *six* and *siy* coordinates for the top left corner of the image. *six* and *siy* are signed integers. If not specified, the image will be positioned at top left corner of the canvas, i.e., at (0, 0).

- `MOV` *id six siy duration*
  Move: This command is used to *translate* the object associated with the identifier *id* to location (*six*, *siy*), where *six* and *siy* can be signed integers. This command will animate the transition of object from its current position to the destination position over the specified *duration*, which is specified in terms of the number of pages.
  Note: *duration* must be strictly greater than 0.

- `SCL` *id fx fy duration*
  Scale: This command is used to *scale* the object associated with identifier *id*. *fx* specifies the scaling factor for the width of the object while *fy* specifies the same for height. *fx* and *fy* both are unsigned floats. The transition is carried out over the specified *duration*, which is specified in terms of the number of pages.
  Note: *duration* must be strictly greater than 0.

- `ROT` *id degrees duration*
  Rotate Anti-clockwise: This command is used to *rotate* the object associated with identifier *id* in *anti-clockwise* direction. *degrees* parameter specifies the degrees by which the object is to be rotated. In order to rotate the image clockwise, specify negative degrees. *degrees* can be a signed float or signed integer. The transition is carried out over the specified *duration*, which is specified in terms of the number of pages.
  Note: *duration* must be strictly greater than 0.

- `ALP` *id alpha duration*
  Alpha: This command is used to change the transparency/alpha channel of the object specified with *id*. *alpha* is an integer between 0 and 255, both inclusive. The transition is carried out over the specified *duration*, which is specified in terms of the number of pages.
  Note: *duration* must be strictly greater than 0.

- `WT` *duration*
  Wait: This command is used to make FLIPER wait on a certain frame for the provided *duration*, which is specified in terms of the number of pages.
  Note: *duration* must be strictly greater than 0.

- `DEL` *id*
  Delete: This command is used to delete an object from the canvas. The object is specified with *id*.

- `BLKB` and `BLKE`
  Block Begin and *Block End*: These commands are used to start and end a block in FLIPER respectively. A block can contain multiple transitions that are run simultaneously. All transitions can have different duration. Transitions which operate on different objects can also be packed into a single block and multiple objects will be animated at the same time. Please refer to the Newton and Tree example. The transitions are applied on each frame in the order they are specified in the source file.
  Note: Blocks cannot be nested, i.e., it is not possible to have blocks inside blocks.
  Note: `DEL` and `WT` are "eager" commands in that the actions will be **performed immediately** upon encounter of the command and not after the block is over.

- *Comments*: All characters following a pound sign(#) are treated as a comment and ignored by FLIPER.
- For the technical details and the actual structure of the grammar, please refer to `grammar.lark`

2 Installation

- FLIPER is written in python. FLIPER relies on 3 libraries to run. They are as follows
  - `lark-parser` - Lark is a parsing toolkit for Python, built with a focus on ergonomics, performance and modularity.
  - `pillow` - Pillow is the friendly PIL fork by Alex Clark and Contributors. PIL is the Python Imaging Library by Fredrik Lundh and Contributors.
  - `moviepy` - Video editing with Python
- You can install all the above dependencies the following command `pip install -r requirements.txt`. I recommend that you create a virtual environment to run the project, in order to eliminate interference from other packages or from different versions of same packages.

3 Interface

`python fliper.py file [(-o | --out) output_file_name] [-fps frames_per_second]`

- `-o | --out` - This option is used to specify the name for the output file. Please note that based on the extension of the output file, the encoding applied by FLIPER will differ.
- `-fps` - This option is to specify the number of frames that will be played per second.
- Conventionally, FLIPER files end with an extension of `.fb`.

4 Examples

FLIPER combines declarative style of coding with procedural flow control. Let us walk through an example which scales down an image to half of its size.

```
CNV 700 700
IMG "img.png" "A" 40 40
SCL "A" 0.5 0.5 90
```

- `CNV 700 700`
  This line creates a *canvas*, an area where the objects will be drawn. The width and height of the canvas are both specified as 700 pixels.
- `IMG "img.png" "A" 40 40`
  This line loads an image with name `img.png` in the memory and associates the *id* `A` with it so that it can be referred to later on in the program. The command also tells FLIPER to position the top left corner of the image at the coordinates (40, 40).
- `SCL "A" 0.5 0.5 90`
  This line tells FLIPER to create a *transition* where the width and height of image associated with *id* `A` are both scaled by 0.5. That essentially halves the width and height of the image. The last argument 90 tells FLIPER to distribute the transition over 90 frames, progressively shrinking the size with each frame.

That's pretty much it. Given the declarative nature of commands, it is very easy to declare transitions and let FLIPER take care of low level details. The other examples bundled with this package are `scale.fb`, `translate.fb` and the **most complicated** `newton.fb`.

5 Extensibility and Future Scope

- FLIPER uses an assembly-like syntax which provides the most flexibility that there is in terms of extending and adding new features.
- Most Domain Specific Languages (DSLs) like FLIPER are aimed at solving very specific and common use cases faced by the people working in the domain.
- To that end, FLIPER is a very small language containing only the most basic operations.
- There are many areas which I wanted to expand upon but could not because of time constraints.
- The declarative nature of FLIPER can sometimes be really prohibitive when trying to do something which the designer did not intend the target audience to do often. The most flexible and most fine grained control over the primitives of language can be provided by providing the ability to add python code snippets to the FLIPER source files, which can be referred to and executed during compilation.
- However, without going to that extreme end, there are many features which can be added which will likely be used by many users. Prime examples of those are loops and branching/conditional flow in FLIPER. Next up, adding subroutine support will make it very convenient to reuse code without duplicating every transition or block of transitions.
- Some basic features which can be added without doing major changes and which may be frequently required are rotation and scaling based on some provided pivot point. That should be a straightforward extension to the language which will benefit a large set of use cases.
- In terms of primitives, the language should not rely on images exclusively. I think FLIPER should offer a set of primitives for creating and manipulating basic shapes such as arcs, circles and polygons.
- In terms of transitions, the two best that I would definitely like to see in a language like FLIPER would be the ability to make an object follow a path described by a set of points, and a general routine to transform/morph a geometric primitive into another.

6 Time efforts

- Total time taken for implementing FLIPER, including finding and deciding on the set of tools to use, designing the grammar and implementing the compiler is about ***14*** hours.