Contents

Running components

In this section, we demonstrate how to run the components of the Civilization postgame generation pipeline in order to generate the postgame video manually. The next section will be about automation.

We refer to earlier sections for an in-depth overview of how the different components are configured.

Building the namespace

Let's build the namespace from the `src` directory in the root of this project/repository:

```
> viash ns build \
+   -n civ6_save_renderer \
+   -s ../src \
+   -p docker
Exporting ../src/civ6_save_renderer/combine_plots/ (civ6_save_renderer) =docker=> target/docker/civ
Exporting ../src/civ6_save_renderer/convert_plot/ (civ6_save_renderer) =docker=> target/docker/civ6
Exporting ../src/civ6_save_renderer/parse_header/ (civ6_save_renderer) =docker=> target/docker/civ6
Exporting ../src/civ6_save_renderer/parse_map/ (civ6_save_renderer) =docker=> target/docker/civ6_sa
Exporting ../src/civ6_save_renderer/plot_map/ (civ6_save_renderer) =docker=> target/docker/civ6_sav
```

The result is stored under `target/docker` because we chose to only build the `docker` platform executables.

We have to run the *setup* for the containers that are not just available on Docker Hub. This can be done in one go by using the following CLI:

```
> viash ns build \
+   -n civ6_save_renderer \
+   -s ../src \
+   -p docker \
+   --setup > /dev/null
```

Or, we can run them one-by-one.

All steps

We took 2 save games from the example data (a very limited set) under `data/` in the root of this repository and stored it locally under `data/`:

```
> ls data/
AutoSave_0158.Civ6Save
AutoSave_0159.Civ6Save
```

Intermediate files will be stored under `temp/`, results will go under `output/`.

`parse_header`

We start by looking at the syntax of `parse_header` to refresh our memory:

```
> target/docker/civ6_save_renderer/parse_header/parse_header -h
Extract game settings from a Civ6 save file as a yaml.

Options:
    -i file, --input=file
        type: file, required parameter, default: save.Civ6Save
        A Civ6 save file.

    -o file, --output=file
        type: file, required parameter, default: output.yaml
        Path to store the output YAML at.
```

We run the component (in its `docker` platform variant) on the two save games:

```
> target/docker/civ6_save_renderer/parse_header/parse_header \
+   --input data/AutoSave_0158.Civ6Save \
+   --output temp/0158.yaml
```

and

```
> target/docker/civ6_save_renderer/parse_header/parse_header \
+   --input data/AutoSave_0159.Civ6Save \
+   --output temp/0159.yaml
```

The result is two YAML files:

```
> ls temp/
0158.pdf
0158.png
0158.tsv
0158.yaml
0159.pdf
0159.png
0159.tsv
0159.yaml
```

`parse_map`

Likewise, we run the `parse_map` component. This component uses a node.js library under the hood, but we need not know that in order to run the component:

```
> target/docker/civ6_save_renderer/parse_map/parse_map -h
Extract map information from a Civ6 save file as a tsv.

Options:
    -i file, --input=file
        type: file, required parameter, default: save.Civ6Save
        A Civ6 save file.

    -o file, --output=file
        type: file, required parameter, default: output.tsv
        Path to store the output TSV file at.
```

We run the component (in its `docker` platform variant) on the two save games:

```
> target/docker/civ6_save_renderer/parse_map/parse_map \
+   --input data/AutoSave_0158.Civ6Save \
+   --output temp/0158.tsv
(node:9) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues
(Use `node --trace-deprecation ...` to show where the warning was created)
```

and

```
> target/docker/civ6_save_renderer/parse_map/parse_map \
+   --input data/AutoSave_0159.Civ6Save \
+   --output temp/0159.tsv
(node:9) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues
(Use `node --trace-deprecation ...` to show where the warning was created)
```

The result is two CSV files next to the already created YAML files:

```
> ls temp/
0158.pdf
0158.png
0158.tsv
```

```
0158.yaml
0159.pdf
0159.png
0159.tsv
0159.yaml
```

`plot_map`

Using both the YAML and CSV files above, we can plot a map for each stage in the game process. We use the `plot_map` to achieve this, first the help:

```
> target/docker/civ6_save_renderer/plot_map/plot_map -h
Use the settings yaml and the map tsv to generate a plot (as PDF).

Options:
    -y file, --yaml=file
        type: file, required parameter, default: header.yaml
        A YAML file containing civ6 game settings information.

    -t file, --tsv=file
        type: file, required parameter, default: map.tsv
        A TSV file containing civ6 map information.

    -o file, --output=file
        type: file, required parameter, default: output.pdf
        Path to store the output PDF file at.
```

We run the component (in its `docker` platform variant) on the two save games:

```
> target/docker/civ6_save_renderer/plot_map/plot_map \
+   --tsv temp/0158.tsv --yaml temp/0158.yaml \
+   --output temp/0158.pdf
── Attaching packages ─────────────────────────────────── tidyverse 1.3.0 ──
✔ ggplot2 3.3.3     ✔ purrr   0.3.4
✔ tibble  3.0.6     ✔ dplyr   1.0.4
✔ tidyr   1.1.2     ✔ stringr 1.4.0
✔ readr   1.4.0     ✔ forcats 0.5.0
── Conflicts ────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

and

```
> target/docker/civ6_save_renderer/plot_map/plot_map \
+   --tsv temp/0159.tsv --yaml temp/0159.yaml \
+   --output temp/0159.pdf
── Attaching packages ─────────────────────────────────── tidyverse 1.3.0 ──
✔ ggplot2 3.3.3     ✔ purrr   0.3.4
```

```
✔ tibble  3.0.6     ✔ dplyr   1.0.4
✔ tidyr   1.1.2     ✔ stringr 1.4.0
✔ readr   1.4.0     ✔ forcats 0.5.0
── Conflicts ──────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

The result is two PDF files:

```
> ls temp/
0158.pdf
0158.png
0158.tsv
0158.yaml
0159.pdf
0159.png
0159.tsv
0159.yaml
```

`convert_plot`

It's now time to convert our PDF version of the plot to PNG format. We do this using the `convert_plot` component:

```
> target/docker/civ6_save_renderer/convert_plot/convert_plot -h
Convert a plot from pdf to png.

Options:
    -i file, --input=file
        type: file, required parameter, default: input.pdf
        A PDF input file.

    -o file, --output=file
        type: file, required parameter, default: output.png
        Output path.
```

And run the tool on the 2 PDF map files we have:

```
> target/docker/civ6_save_renderer/convert_plot/convert_plot \
+   --input temp/0158.pdf \
+   --output temp/0158.png
```

and

```
> target/docker/civ6_save_renderer/convert_plot/convert_plot \
+   --input temp/0159.pdf \
+   --output temp/0159.png
```

It's as simple as that.

```
> ls temp/
0158.pdf
```

```
0158.png
0158.tsv
0158.yaml
0159.pdf
0159.png
0159.tsv
0159.yaml
```

`combine_plots`

The last step is to render the video based on the (only 2) PNG image files:

```
> target/docker/civ6_save_renderer/combine_plots/combine_plots -h
Combine multiple images into a movie using ffmpeg.

Options:
    -i file1:file2:..., --input=file1:file2:...
        type: file, required parameter, multiple values allowed, default: plot1.png:plot2.png
        A list of images.

    -o file, --output=file
        type: file, required parameter, default: output.webm
        A path to output the movie to.

    -f integer, --framerate=integer
        type: integer, default: 4
        Number of frames per second.
```

```
> target/docker/civ6_save_renderer/combine_plots/combine_plots \
+   -i temp/0158.png:temp/0159.png \
+   -o output/video.webm \
+   -f 1
ffmpeg version 4.1 Copyright (c) 2000-2018 the FFmpeg developers
  built with gcc 5.4.0 (Ubuntu 5.4.0-6ubuntu1~16.04.11) 20160609
  configuration: --disable-debug --disable-doc --disable-ffplay --enable-shared --enable-avresample
  libavutil      56. 22.100 / 56. 22.100
  libavcodec     58. 35.100 / 58. 35.100
  libavformat    58. 20.100 / 58. 20.100
  libavdevice    58.  5.100 / 58.  5.100
  libavfilter     7. 40.101 /  7. 40.101
  libavresample   4.  0.  0 /  4.  0.  0
  libswscale      5.  3.100 /  5.  3.100
  libswresample   3.  3.100 /  3.  3.100
  libpostproc    55.  3.100 / 55.  3.100
Input #0, png_pipe, from 'concat:/viash_automount<...>/workspace/di/viash_workshop_1/150-RunningCo
  Duration: N/A, bitrate: N/A
    Stream #0:0: Video: png, rgba64be(pc), 1728x936 [SAR 72:72 DAR 24:13], 1 tbr, 1 tbn, 1 tbc
Stream mapping:
  Stream #0:0 -> #0:0 (png (native) -> vp9 (libvpx-vp9))
```

```
Press [q] to stop, [?] for help
[libvpx-vp9 @ 0xd91440] v1.8.0
Output #0, webm, to '/viash_automount<...>/workspace/di/viash_workshop_1/150-RunningComponents/outp
  Metadata:
    encoder         : Lavf58.20.100
    Stream #0:0: Video: vp9 (libvpx-vp9), yuva420p, 1728x936 [SAR 1:1 DAR 24:13], q=-1--1, 200 kb/s
    Metadata:
      encoder         : Lavc58.35.100 libvpx-vp9
    Side data:
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: -1
frame=    2 fps=2.0 q=0.0 Lsize=     137kB time=00:00:01.00 bitrate=1121.0kbits/s speed=0.984x
video:136kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: 0.707957%
```

This concludes the manual approach to running the scripts. In the next section, we will automate this by means of a bash script.