>Remember

By Hugo Labrande
Issue #8 : Browser-based graphical text adventures

This month's article is another technical article, outlining a way for you to make retro graphical text adventures in a browser. Adventuron is already fairly established in that niche, but the path I'll be talking about here uses Inform 6, and its very powerful parser, and has the ability to sprinkle in some JavaScript if you want more advanced features. It is one of the only ways to have an Inform 6 game with pictures, and while there is very little chance it'll ever run on a retro computer, the fact that it can run in a browser makes it very easy to share. Highly recommended for your own games, or if you want to make a browser remake of an old game!

Quick Vorple introduction

Vorple is a very interesting tool created by Juhana Leinonen to augment Inform's capabilities. The homepage is here:

https://vorple-if.com/

The basic idea is that there is a regular interpreter in a browser, that can also interpret JavaScript/CSS commands started within the story file with a special function. This means you can write Inform code that controls the game logic and is able to communicate with the browser, tell it to change the layout, execute a complex function, or even recover information from Wikipedia or any other server! There are a few functions built in the Vorple libraries, but you can do anything if you know JavaScript. If you don't, that's fine: today's tutorial doesn't require you to know anything about JS.

(Note that, for now, this tutorial doesn't work with PunyInform: Vorple is only compatible with Glulx, and PunyInform is not compatible with Glulx. It's not hard to go from PunyInform to the standard I6 library and vice-versa: there's only a few adaptations that you need to do to your code, and they are detailed in the PunyInform manual. Sorry!)

Vorple requires a little bit of setting up, but not too much; ultimately, it's just a set of extensions and a custom interpreter. First, start by downloading the contents of this Github repository:

github.com/vorple/inform6

The ".h" files are the extensions; we will only require "vorple.h" (core library), "vorple-multimedia.h" (pictures), and "vorple-status-line.h". They need to be included in your Inform source file, as follows:

```
Include "vorple.h";
Include "Parser";
Include "VerbLib";
Include "vorple-multimedia.h";
Include "vorple-status-line.h";
```

Another thing you have to do is edit the "play.html" file; this is the one that runs your game in the javascript interpreter. Edit it so the line `story: "test.ulx"` refers to your game. For now Vorple is only compatible with Glulx, which means you have to specify the "-G" option when compiling; the result should be a "game.ulx" file. Finally, note that there is a "resources" folder; we will look into that soon.

One major thing to note about Vorple is that double-clicking on the html file will not launch the game correctly. (You'll get a server error!) In order to run, the html file needs to be fed through an HTTP server. There are several solutions to this: zip your whole folder and upload it to your itch.io account; make the whole folder sync to your Dropbox/OneDrive and access it through your Dropbox/OneDrive account in a browser; or run a local HTTP server on your computer ("`python3 -m http.server`" in a terminal, for instance).

Once all of this is setup, you should hopefully get a successful compilation, and manage to have your game run in the browser. Let's add pictures!

Integrating pictures

The basic idea to integrate pictures above the scrolling text, like in a classic graphical text adventure, is to put the image in the status line, which by default is an element sitting at the top of the window and under which the scrolling text disappears. We thus need to insert the picture of the location in the status line every time the status line is drawn.

The way Vorple handles status line refreshes is by looking for, and executing, any code attached to this event; by default, this is just the regular status line. To attach code to be executed every time the status line is refreshed, you need to put it in an object, and put this object in the "StatusLineRulebook"; each turn, Inform will look at all the objects in that rulebook and run their code. Our particular code will need to switch focus to the center of the status bar, do its thing, then give back focus. Something like this:

```
Object MyStatusLineRule "" StatusLineRulebook,
    with description [;
        ! erase the contents in the old status line
        VorpleStatusLineClear();

        ! Put the "cursor" at the center of the status line
        VorpleSetOutputFocus("status-line-middle");

        ! Do something here

        ! Put the cursor back in the text zone
        VorpleSetOutputFocusMainWindow();
];
```

Let's leave this code for now, and look into how to specify a picture corresponding to each location. Inform 6 is an object-oriented language, so anything we tie to an object should be contained to that object; meaning, the best way is to specify inside the location object which picture should be displayed. I'll create a property named "`imageName`" inside each location, and give the path to the image file; this path should be relative to the base folder for images, which by default (you can edit "play.html" if you want) is "resources/images". My code looks like this:

```
Object aztecPyramid "Aztec Pyramid"
    with description "You climb the last few steps, wipe the sweat off
your brow, and look at the surrounding jungle.",
    imageName "pyramid.png";

Object jungle "In the jungle"
    with description "The humid heat sticks o your clothes.",
    imageName "jungle.png";
```

We can then fill in the blank in the rule we had set up before:

```
Object MyStatusLineRule "" StatusLineRulebook,
    with description [;
        ! erase the contents in the old status line
        VorpleStatusLineClear();
        ! Put the "cursor" at the center of the status line
        VorpleSetOutputFocus("status-line-middle");
        ! Display the location’s image
        VorpleImage(location.imageName, "", IMAGE_CENTERED);
        ! Put the cursor back in the text zone
        VorpleSetOutputFocusMainWindow();
];
```

Note that because of the way the elements are arranged on the page, you might need to pad your text area so no text is ever displayed under the picture. (A symptom of this is if you have just a picture, and no intro text showing.) Adding the command

```
   VorpleExecuteJavaScriptCommand("$('#window0').css('margin-top',
'320px');");
```

at the end of your “Initialise” function should do the trick, though you might have to tweak the pixel value depending on the height of your pictures.

That's it!

And that's enough for today! Should you want to go further, I'll leave the following as exercises:

- make your images looping GIF files (for instance, water dripping from leaves in the jungle) and display them;
- add your own CSS file to the page and change the font to a retro font of your choice, such as “Press Start 2P”
- if you prefer Inform 7, port this code to it (or ask me, I have it somewhere!);
- make imageName a routine that returns a different image name depending on whether it's day or night in your game;
- add more graphical elements to the interface, such as a compass that updates at every room;
- use Electron to create an executable version of your game so it can run on Windows and be sold on Steam (it's not too hard, ask me!).

Vorple is a really great tool to augment text adventures; I hope this month's article will have sparked some interest and make you want to experiment further with it!