IDE: Visual Studio Code.

Recommended Extensions: Javascript Debugger, Cordova Tools

Language: Javascript

Framework: Cordova (<https://cordova.apache.org/>)

Debugged in Browser mode but planning on turning this into an app eventually.

Setup: Download Repo. Install Cordova. (Terminal: npm install -g cordova) Install chrome as your platform in cordova. (Terminal: cordova platform add browser)

To run the program: Terminal: cordova run browser

A window will pop up, and it will say its loading. Default is ten years of simulation. After a few seconds it will complete and turn green. Press Fn F12 to show developer tools and see some of the system logs from the simulation. Press ctrl C twice in VS code to quit, then close the chrome tab. I couldn’t really get the VS code debugger to work, so what I would do is put a breakpoint on line one of VS code then go to debugger tools for the instance in chrome. From there you can set breakpoints and step forwards and inspect whatever you need to.

Files:

Index.css: The css file. Not really important yet.

Index.html: Not really important yet. If you create a new .js file, make sure to reference it before index.js here.

(I suggest for index.js and simulate.js you collapse all the functions or else its gonna be very overwhelming)

Index.js: Handles the world generation. Utility functions first, then globals, then classes, then functions. The actual calls are made at the very bottom in generate world(). The simulation then begins with simulate(days, years), which calls the simulate.js file. Last but not implemented is the playgame() call, which will be simulate copy pasted then edited so that the player can interact with the world. It’s important to make the simulation have a version with or without the player in case we need to do timeskips. It’s also useful because we can simulate the world for a few hundred years to make it feel very alive and ‘lived in’ before the player ever starts.

Simulate.js: the simulation component of the project. Everything in this file has to be inside of the simulate() call or else javascript throws a fit. At the very bottom is the main loop. 100 days per year, 3 phases per day, one goal is acted towards per phase with potentially unlimited actions resulting from that (but usually its only 1 or 2). First all the characters go, then the monsters, and finally the background logic occurs. That’s stuff like birthing new characters, aging people, etc.

Lists.js : a file with a bunch of hardcoded lists that the random number generator uses for deciding names and attributes and stuff. Eventually I would like to host these lists on a server somewhere and make it so that approved contributors can add to it, but that’s way down the road. Most of these values were chosen from whatever google resulted when I searched “list of male names” or “cool fantasy names” etc.

As for what the functions do, its largely clear based on comments and naming conventions. I tried to keep it relatively straightforward, but didn’t always succeed so just ask me questions on confusing bits. Start with Index.js before simulate.js . This code isn’t very well optimized, organized, or written, so if you see ways to improve it without breaking anything go for it.

One last thing to point out before you look at the code: I’ve kept some lists global so that it’s easy to sort through the huge lists of information. Its important to make the distinction that some lists hold literal values, while others hold object references. For example, if I just made a new character, it will say at the end:

characters.push(character);

keywords.push(character.name);

The characters list holds a reference to the actual character, while the keywords list holds a name for the character. This means that If you want to find or set information about a character, you need to loop through the characters list to find them, then loop through their characteristics since they are an object. Here is an example.

        //characters like those they travel with.

        for (char of characters) {

            for (key of char.party){

                for (opi of char.opinions){

                    if (opi.keyword == key && opi.keyword != char.name){

                        if (opi.affinity <80){opi.affinity++;}

                    }

                }

            }

        }

In javascript, for loops use either ‘in’ or ‘of’. ‘in’ is an indexed loop, which would be used if you want to set characters[i] to some value. ‘of’ is a reference loop, which is a little easier to read since you don’t have to mess with array indexing.

Here is what is going on above…

Iterate through the characters array, using the reference char.

For every single character in the character array (the current one being referenced as char), iterate through their current party using the reference ‘key’.

For every single opinion of every single character in the character array, set the reference as opi.

If the character has an opinion on the keyword key and that keyword is not their own name, and if their affinity for that keyword is less than 80, then increment the current characters opinion of that keyword by one.

So basically, go through every single character. Look at who is in their party. Increment the current characters affinity for the people in their party by one. Since a person’s party by default includes themselves, don’t look at that one.

^ this method of finding and setting values is why the program runs slowly but its what I came up with at the time so I just kept on using it. Its easier to read if you look at what is being changed or edited first, then work up the loop chain.

I originally was going to write down a big explanation of every single function but I’ve decided it’d be easier if you just called me and I explained it while you looked at it. So, yeah. 😊