Info MS_DOS javascript

Basic Game Setup Info

Each game has a basic HTML div with the name of the game as ID as well as an a div inside of it with NAMEOFGAME_area as class.

Everything you add to the html code with javascript has to go inside these 2 div"s.

Files Structuring

The game files are split between the GAMENAME.js, GAMENAME_core.js and several other files like movement, AI etc. depending on the game.

Each game has it's own CSS and JS files, as well as it's own img folder to prevent a huge mess of files and functions all over the place. All of these files are placed inside the games/GAMENAME folder in their respective folders.

GAMENAME.js has the OnLoad function, as well as functions that catch keyboard/mouse activity (OnKeyDown, OnKeyUp, OnClick).

the GAMENAME.js file is located in the GAMENAME folder, everything else is located inside GAMENAME/js/

All variables, functions (except for the onLoad, onKeyDown, onKeyUp and onClick), classes and ids use the ID Code of the game in front of them to prevent overlapping names between games.

ex: IDCODE_Move() instead of Move()

HTML & Javascript strings

In php you can combine html and php code by using a . between them, however in javascript this is done by using a +

```
'test' . $var -> 'test' + var
```

JSON html structure

json is used to create the popup a well as any static html inside of the popup (gameover screen, title screen etc.).

each html element has a corresponding object in the json structure

nested elements are added to the children object of the parent

the following object keys are available:

- "tag" defines the type of html element
- "id" defines the element id
- "class" defines the element class(es)
- "text' defines the plaintext content of the element
- "children" is an array of 1 or more element objects
- "svg" is only used for the close button of the popup, it should not be used on other elements

dynamic html elements (grid, enemies, obstacles etc.) should not be generated with the json, since these elements change during the gameplay.

JSON only uses objects {} and arrays [].

Objects contain key : value pairs. An object can never directly be used as a value in a key : value pair.

An array is only allowed to hold other objects. It can also be used as a value in a key : value pair.

Variables and Event Listeners

Example OnLoad variables:

```
this.GamePlaying = 0,
this.Score,
this.Level = 1,
this.Array = []
```

`this.Your_Var` is only used in the onLoad function, in all other functions you put

`var that = window.GAMENAMEObj;` at the top, and call variables with `that.Your_Var`

Onload, OnKeyDown, OnKeyUp and OnClick are event listeners. This means that these 4 functions are executed whenever the corresponding action happens.

- OnLoad executes when the game file is loaded. This only happens once, unless you refresh the webpage or close and restart the browser.
- OnKeyDown executes when you press a key
- OnKeyUp executes when you release a key after pressing it
- OnClick executes when you click inside the game window

Keyboard code names are used with keydown and keyup event listeners and can be found on the following site:

https://keyjs.dev/

variables in javascript work a bit different from php variables, in php `$a = $b` will mean you have 2 seperate variables with the same value

in javascript `that.a = that.b` means that that.b is “connected” to `that.a`. This means that changing the value in `that.b` will also change the value in `that.a`

You can use Copy(varname); function in this system to create a copy of a variable.

Selecting elements

`$("#your_elementid")` tells the code which element has to be selected, this can be done with using an ID or a CLASS

Keep in mind that if you use multiple elements with the same class, you need to loop through the selected elements , or else it won't work since javascript doesn't know which of the 5 elements with that class name you want to use

Javascript/Jquery functions that start with a . (.hasClass() for example) are always used with `$("#your_elementid")`

```
$("#your_elementid").hasClass("classname");
```

List of common functions that start with . :

`.width()` gets the width of the selected element

`.width("10px")` gives the selected element a new width

This works the same with `.height()`

You can also make your width and height the same by using the example below

```
$("#your_elementid").width( $("#your_elementid").height() );
```

`.html('text')` adds everything between () tot the selected element

$("#div").html('text');

`<div id="div"></div>` becomes `<div id="div">text</div>`

Keep in mind that `.html()` replaces any and all html that was previously inside the selected element

`.append('text')` adds everything between () tot the selected element

As opposed to `.html()`, `.append()` does not remove any of the existing html, instead adding to it

```
$("#div").append('text');
$("#div").append("more text');
```

`<div id="div"></div>` becomes `<div id="div">text more text</div>`

`.css()` expects 2 values, the css to be changed, and the new value

If you only give a number to the second value it will automatically add "px" behind it, if you don't want this you will have to use + in order to add another extension to it

```
.css("left', 10 + "%")
```

`parsefloat("100px")` removes "px" from the value, which is usefull for calculations as they will not work with most extensions (also works with "%" and most other extensions)

"100px" * "75px" doesn't work

parseFloat("100px") * parseFloat("75px") does work

"100%" * "75%" doesn't work

parseFloat("100%") * parseFloat("75%") does work

Interval & SetTimeout

that. IDCODE_interval = setInterval(function(){

IDCODE _MovePlayer();

IDCODE _MoveEnemy();

}, 50);

Interval is a repeating event that calls and/or executes everything within the {} every X (50 in this example) ticks/milliseconds

Always bind/add setInterval to a variable, otherwise it will be difficult to stop/remove it without removing every other interval as well

`clearInterval(that.IDCODE_interval);` this function removes the interval that is given as a parameter, thus stopping the repeated calls/executions of everything inside it's {}

This function can be used to stop the game for running when you get a game over screen or when you pause the game

Not giving this function a parameter will cause it to remove all intervals that are currently active

```
setTimeout(function() {
	IDCODE_MovePlayer();
	IDCODE_MoveEnemy();
}, 50);
```

As opposed to intervals, setTimeout will only call/execute the code inside {} once, so you don't need to juggle multiple intervals around just for that 1 thing that only happens in specific situations (like a player death animation)

Random Number Generator

Getting a random number in javascript/jquery works a little bit different from php

Below is a piece of code that will generate a random number between 1 and 10

```
var rng = (Math.floor(Math.random() * 10) + 1);
```