



DESENVOLUPAMENT D'UN JOC EN HTML5 I JAVASCRIPT

TITULACIÓ: ENGINYERÍA TÉCNICA EN INFORMÀTICA DE SISTEMES

AUTORS: MARIO RINA BLANC
DIRECTORS: JORDI DUCH GAVALDÀ

DATA: SEPTIEMBRE / 2011.

Indice

1- Objetivos	pag 4
2- Especificaciones	pag 5
2.1 - HTML5: Historia y novedades	pag 5
2.1.1- Novedades	pag 6
2.1.2- Etiquetas	pag 7
2.1.3- Lista de etiquetas y función que realizan	pag 10
3- Diseño y desarrollo	pag 14
3.1 - Clases, estructuras y ficheros	pag 15
3.1.1- Estructura Inicio	pag 15
3.1.2- Clase Square	pag 16
3.1.3- Clase ImageData	pag 17
3.1.4- Clase Character	pag 18
3.1.5- Clase ImageSet	pag 20
3.1.6- Estructura Pausamos	pag 21
3.1.7- Clase pantGame	pag 22
3.1.8- Estructura QueSeHace	pag 24
3.1.9- Estructura Selecciones	pag 25
3.1.10- Estructura Vars	pag 25
3.1.11- Estructura Volver	pag 25
3.1.12- Clase pantHelp	pag 26
3.1.13- Clase pantIni	pag 27
3.1.14- Clase pantPause	pag 28
3.1.15- Clase pantOut	pag 29
3.1.16- Estructura Tecles	pag 30
3.1.17- Estructura Pulsar	pag 31
3.1.18- Código HTML5 document	pag 32
3.1.19- Clase mygame	pag 33
3.1.20- Archivos obtenidos para poner sonidos al juego	pag 35
3.1.21- Fichero menu.js	pag 36
3.1.22- Fichero prueba.html	pag 36

3.2- Diseño de clases completo	pag 37
4- Evaluaciones (Pruebas para que todo funcione)	pag 38
5- Conclusiones	pag 43
6- Recursos utilizados	pag 44
6.1- Bibliografía	pag 44
6.2- Páginas web	pag 44
6.3-Programas utilizados	pag 44
7- Anexos	pag 45
Anexo 1: NAVEGADORES	pag 45
Anexo 2: INSTRUCCIONES	pag 46
8- Resúmenes	pag 54
8.1- Resumen en Español	pag 54
8.2- Resumen en Catalán	pag 55
8.3- Resumen en Inglés	pag 56

1- Objetivos del proyecto

El proyecto tiene como objetivo realizar un videojuego utilizando JavaScript y HTML5, el cual aporta novedades respecto a las otras versiones.

Otro de los objetivos que se pretende conseguir con este proyecto es que el videojuego sea multiplataforma, es decir, que funcione en los diferentes navegadores Web, como Google Chrome, Safari, Mozilla firefox,...

Se quiere implementar un proyecto que contenga, como ya se ha dicho, las novedades que el nuevo estándar del HTML5, las cuales se entrelazan con el lenguaje javascript.

2- Especificaciones del proyecto

Lo que se aspira a realizar es, como ya se ha dicho en los Objetivos del proyecto, un juego desarrollado en javascript que incorpore las novedades que trae consigo el lenguaje HTML5.

Se pretende realizar un videojuego que utilice las diferentes herramientas que proporciona el lenguaje web HTML5. Con ello, los usuarios se podrán mover e interactuar en el videojuego gracias a los diferentes eventos, como el ratón o el teclado.

También, a través de una clase principal, que sólo contenga elementos básicos y muy genéricos, se distribuya por todas las clases, es decir, que todas las clases que se tengan que emplear, tengan parte de los elementos base, a fin de que quede todo más fácil de entender.

A fin de conseguir lo mencionado, es decir, que todo quede simple y bien expresado, cada grupo de acciones deberán ir en su clase, y no en otra, ya que sino puede dar lugar a confusiones y errores.

Veamos a continuación un poco de la historia del lenguaje HTML5 y sus novedades, para entender más tarde cómo se ha realizado el proyecto, haciendo uso de éste lenguaje.

2.1 - HTML5: HISTORIA Y NOVEDADES

Hoy en día hay pocas páginas web que se basen únicamente en HTML. Muchas personas utilizan CSS para definir el aspecto de la página, algún script en Javascript, videos en formatos o Flash para realizar animaciones. Para beneficiarse, existen etiquetas que se han creado al paso, según se necesitaban.

Así pues, más de 10 años después de la última especificación de HTML, resulta importante para el futuro la creación de un nuevo estándar que recoja y solucione de alguna forma las necesidades de los desarrolladores que se han creado durante este tiempo.

HTML5 no se dedica únicamente a crear nuevas etiquetas, atributos y eliminar las marcas que están en desuso o no se utilizan correctamente, sino que va un paso más lejos.

Entre las nuevas versiones de diversas especificaciones, se encuentran:

- HTML 4.
- XHTML 1.
- DOM Nivel 2 (Document Object Model).

También pretende proporcionar una plataforma con la de realizar aplicaciones parecidas a las aplicaciones de escritorio, donde la ejecución dentro del navegador no implique una falta de recursos o facilidades para resolver diferentes necesidades de los desarrolladores. Para ello se crean unas APIs que permiten trabajar con cualquier elemento de la página y realizar acciones que hasta ahora se tenían que realizar por medio de tecnologías accesorias.

Las APIs, que son implementadas por los diferentes navegadores, se están documentando con minuciosidad, para que todos los Browsers las soportan tal como se han diseñado. Esto se hace para que no ocurra lo que pasaba en el pasado, ya que era frecuente que se accediera a sitios webs que no eran compatibles con el navegador preferido de cada usuario.

Aunque todavía no ha salido HTML5 como especificación de implementación (se preveé que esté listo en 2012), algunos navegadores ya implementan características del lenguaje. Esto ocurre porque HTML5 está formado por diferentes módulos, los cuales hay algunos que ya están listos para ser implementados, pero otros todavía están como borradores.

2.1.1- NOVEDADES

No solo trata de incorporar nuevas etiquetas o eliminar otras, sino que hay mejoras en diversas áreas para las cuales se requería utilizar otras tecnologías.

- *Estructura del cuerpo*: La mayoría de las webs tienen un formato común, formado por diferentes elementos, como la cabecera, pie, navegadores... HTML5 agrupa todas estas partes en nuevas etiquetas que representarán cada una de las partes de una página.

- *Etiquetas para contenido específico*: Así como antes se utilizaba una sola etiqueta para incorporar diferentes tipos de contenido (como Flash o vídeo), ahora se utilizan etiquetas para cada tipo de contenido, como audio, vídeo,...

- *Canvas*: Es un componente que permite dibujar, por medio de las funciones de una API todo tipo de formas en la pagina, las cuales podrán estar animadas y responder a la interacción del usuario. Fue desarrollado inicialmente por Apple para el navegador Safari y después fue utilizado por la organizacion WHATWG para incorporarlo a HTML5. Ha sido adoptado por Firefox y Opera. Chrome, como utiliza el mismo tipo de renderizado que Safari, soporta el elemento Canvas. La última version de IE (IE8), no soporta canvas con funciones nativas, pero hay formas de poder utilizar el elemento.

- *Bases de datos locales*: El navegador permite el uso de una base de datos local, con la cual se podrá trabajar en una página web por medio de un cliente a través de una API. Es parecido a las cookies, pero están pensadas para almacenar grandes cantidades de información.

- *Web workers*: Procesos que requieren tiempo de procesamiento por parte del navegador, pero se pueden realizar en un segundo plano, para que el usuario no se tenga que esperar a que termine para usar la página.

- *Aplicaciones web Offline*: Existe otra API para el trabajo con aplicaciones web, las cuales se pueden desarrollar de modo que funcionen también en local y sin estar conectados a Internet.
- *Geolocalización*: Las páginas web se podrán localizar geográficamente por medio de un API que lo permita.
- *Nuevas APIs para interfaz de usuario*: Temas tan utilizados como el “drag & drop” en las interfaces serán incorporadas a HTML5 por medio de un API.
- *Fin de las etiquetas de presentación*: Todas las etiquetas que tienen que ver con la presentación del documento serán eliminadas. La responsabilidad de definir el aspecto correrá a cargo de CSS.

2.1.2- ETIQUETAS:

Respecto a las etiquetas, con la llegada de CSS muchas de las etiquetas que afectaban a la presentación de elementos en pantalla cayeron en desuso.

ETIQUETAS EN DESUSO: (existe una forma de hacerlo con CSS):

- font
- center
- strike
- basefont
- big
- s
- tt
- u

Debido a que HTML5 elimina el uso de frames por razones de usabilidad (no pasa eso con IFRAME), las etiquetas vinculadas que se eliminan son:

- frame
- frameset
- noframes

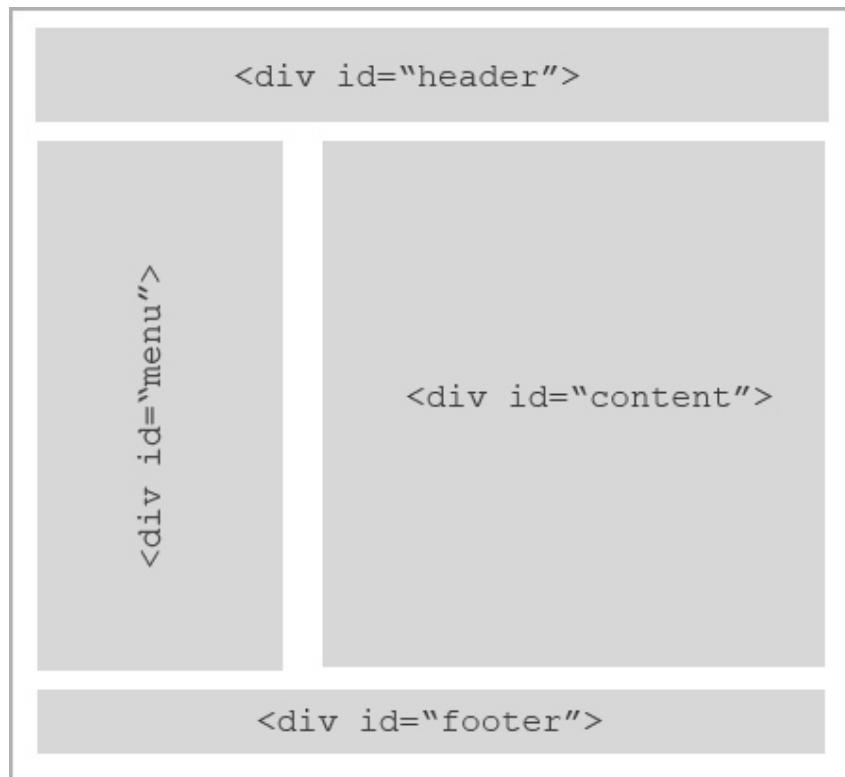
Etiquetas que se han eliminado debido a que creaban confusión o hacían lo mismo que otras son:

- acronym
- applet
- isindex
- dir

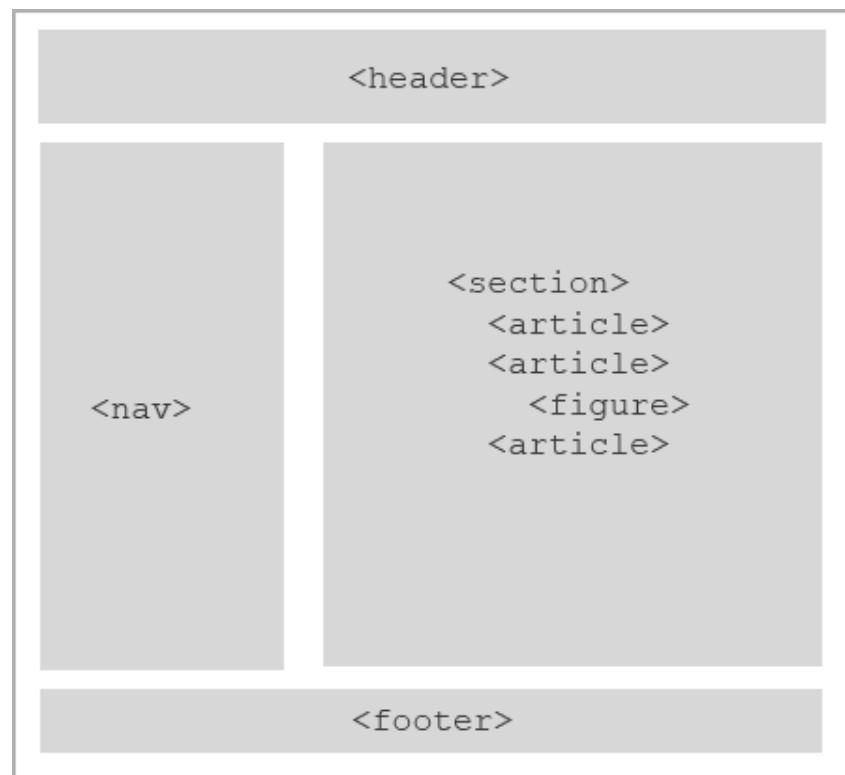
Las etiquetas que aparecen nuevas son las siguientes:

- section
- article
- aside
- hgroup
- header
- footer
- nav
- dialog
- figure
- video
- audio
- embed
- mark
- progress
- meter
- time
- ruby (rt y rp)
- canvas
- command
- details
- datalist
- keygen
- output
- datagrid
- menu

Un pequeño ejemplo de las diferencias entre el HTML4 y el HTML5 reside en lo siguiente. Mientras que en HTML4 se hubiese realizado algo así:



en HTML5 se puede hacer más explícito el contenido de las secciones en lugar de usar identificadores para los “div”.



2.1.3- Lista de etiquetas y función que realizan:

Page 1 of 4

Quick Reference Guide

FREE

HTML 5

Tag	Info	V	Attributes*
<!-- -->	comment	4 / 5	none
<!DOCTYPE>	document type	4 / 5	none
<a>	hyperlink	4 / 5	href hreflang media ping rel target type
<abbr>	abbreviation	4 / 5	global attributes**
<acronym>	acronym	4	-
<address>	address element	4 / 5	global attributes**
<applet>	applet	4	-
<area>	area inside an image map	4 / 5	alt coords href hreflang media ping rel shape target type
<article>	article	5	global attributes**
<aside>	outside the main flow of the narrative	5	global attributes**
<audio>	sound content	5	autobuffer autoplay controls loop src
	bold text	4 / 5	global attributes**
<base>	base URL for all the page links	4 / 5	href target
<basefont>	Base font for the document	4	-
<bb>	invoked user agent command	5	type
<bdo>	direction of text display	4 / 5	dir
<big>	big text	4	-
<blockquote>	long quotation	4 / 5	cite
<body>	body element	4 / 5	global attributes**
 	inserts a single line break	4 / 5	global attributes**
<button>	push button	4 / 5	autofocus disabled form formaction formenctype formmethod formnovalidate formtarget name type value
<canvas>	Graphic area	5	height width
<caption>	table caption	4 / 5	global attributes**
<center>	centered text	4	-
<cite>	citation	4 / 5	global attributes**
<code>	computer code text	4 / 5	global attributes**
<col>	attributes for table columns	4 / 5	span
<colgroup>	groups of table columns	4 / 5	span
<command>	command button	5	checked default disabled hidden icon label radiogroup type
<datagrid>	data in a tree, list or tabular	5	disabled
<datalist>	dropdown list	5	global attributes**
<dd>	definition description	4 / 5	global attributes**
	deleted text	4 / 5	cite datetime
<details>	details of an element	5	open
<dialog>	dialog (conversation)	5	global attributes**
<dir>	directory list	4	-
<div>	section in a document	4 / 5	global attributes**
<dfn>	definition term	4 / 5	title
<dl>	definition list	4 / 5	global attributes**
<dt>	definition term	4 / 5	global attributes**
	emphasized text	4 / 5	global attributes**
<embed>	external interactive content or plugin	5	height src type width
<fieldset>	fieldset	4 / 5	disabled form name
<figure>	group of media content, and their caption	5	global attributes**
	text font, size, and color	4	-
<footer>	footer for a section or page	5	global attributes**
<form>	form	4 / 5	action data replace accept accept-charset enctype method target
<frame>	sub window	4	-
<frameset>	set of frames	4	-
<h1> to <h6>	header 1 to header 6	4 / 5	global attributes**
<head>	information about the document	4 / 5	none
<header>	header for a section or page	5	global attributes**
<hgroup>	heading section	5	global attributes**
<hr>	horizontal rule	4 / 5	global attributes**
<html>	html document	4 / 5	manifest
<i>	italic text	4 / 5	global attributes**
<iframe>	inline sub window (frame)	4 / 5	src name sandbox seamless width height
	image	4 / 5	alt src height ismap usemap width
<input>	input field	4 / 5	accept alt autocomplete autofocus checked disabled form formaction formenctype formmethod formnovalidate formtarget height list max maxlength min multiple name pattern placeholder readonly required size src step type value width
<ins>	inserted text	4 / 5	cite datetime
<isindex>	single-line input field	4	-
<kbd>	keyboard text	4 / 5	global attributes**
<label>	label for a form control	4 / 5	for
<legend>	fieldset title	4 / 5	global attributes**
	list item	4 / 5	value
<link>	resource reference	4 / 5	href rel media hreflang type sizes
<mark>	marked text	5	global attributes**
<map>	image map	4 / 5	id
<menu>	menu list	4 / 5	label type
<meta>	meta information	4 / 5	charset content http-equiv name
<meter>	measurement within a predefined range	5	high low max min optimum value
<nav>	navigation links	5	global attributes**
<noframes>	noframe section	4	-
<noscript>	noscript section	4 / 5	none
<object>	embedded object	4 / 5	data height type usemap width object

* Attributes: Lists attributes specific to that tag. Deprecated (HTML4 only) attributes are not listed
 ** Global Attributes: class | contenteditable | contextmenu | dir | draggable | id | irrelevant | lang | ref | registrationmark | tabindex | template | title

V = Which version of HTML is this tag valid for

HTML 5 - extended

Tag	Info	Attributes
<!-- -->	comment: comments are displayed in code only. Tag contents are not rendered in the browser	none
<!DOCTYPE>	document type: defines which specification the document follows	none
<a>	anchor: used to provide a link to another web resource href: destination resource of the hyperlink hreflang: gives the language of the linked resource media: describes for which media the target document was designed ping: gives the URLs of the resources that are interested in being notified if the user follows the hyperlink rel: relationship between the document containing the hyperlink and the destination resource [alternate archives author bookmark contact external feed first help icon index last license next nofollow noreferer pingback prefetch prev search stylesheet sidebar tag up] target: gives the name of the browsing context that will be used [_blank _parent _self _top] type: gives the MIME type of the linked resource	
<abbr>	abbreviation: an abbreviation or acronym, optionally with its expansion	global attributes**
<address>	address element: represents the contact information for its nearest article or body element ancestor	global attributes**
<area>	area: either a hyperlink with some text and a corresponding area on an image map, or a dead area on an image map alt: alternate text for the area coords: coordinates for the clickable area href: destination resource of the hyperlink hreflang: gives the language of the linked resource media: describes for which media the target document was designed ping: gives the URLs of the resources that are interested in being notified if the user follows the hyperlink rel: relationship between the document containing the hyperlink and the destination resource [alternate archives author bookmark contact external feed first help icon index last license next nofollow noreferer pingback prefetch prev search stylesheet sidebar tag up] shape: defines the shape of the area [default rect rectangle circ circle poly polygon] target: gives the name of the browsing context that will be used [_blank _parent _self _top] type: gives the MIME type of the linked resource	
<article>	article element: a section of a page that consists of a composition that forms an independent part of a document, page, or site	global attributes**
<aside>	aside element: a section of a page that consists of content that is tangentially related to the content around the aside element, and which could be considered separate from that content	global attributes**
<audio>	sound content: represents a sound or audio stream autobuffer: determines if the audio will be buffered [autobuffer] autoplay: determine if the audio will automatically play [autoplay] controls: indicates that the author has not provided a scripted controller and would like the user agent to provide its own set of controls [controls] loop: sets whether the audio will start once the end is reached [loop] src: URL of the audio to play	
	bold text: creates text that will be made bold	global attributes**
<base>	base element: base URL for all the page links	href: URL to use as the base URL for links in the page target: sets the base target for links in the page [_blank _parent _self _top]
<bb>	browser button: a user agent command that the user can invoke	type: indicates the kind of command [makeapp]
<bdo>	bdo element: represents explicit text directionality formatting control for its children	dir: direction override [ltr rtl]
<blockquote>	block quote element: a section that is quoted from another source	cite: URL of the origin of the quote
<body>	body element: main content of the document	global attributes**
 	break: inserts a single line break	global attributes**
<button>	button: a button page element	autofocus: indicate that a control is to be focused as soon as the page is loaded [autofocus] disabled: prevents the button from being pressed [disabled] form: used to explicitly associate the button element with its form owner formaction: URL that specifies a form processing agent formenctype: specifies the content type used to submit the form to the server [application/x-www-form-urlencoded multipart/form-data text/plain] formmethod: which HTTP method will be used to submit the forms data [get post put delete] formnovalidate: indicate whether the form is to be validated before submission [formnovalidate] formtarget: the target when the form is submitted [_blank _parent _self _top] name: elements name type: controls the behavior of the button when it is activated [submit reset button] value: gives the element's value for the purposes of form submission
<canvas>	canvas element: a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, or other visual images on the fly	height: height of the canvas in pixels - default is 150 width: width of the canvas in pixels - default is 300
<caption>	table caption: the title of the table that is its parent, if it has a parent and that is a table element.	global attributes**
<cite>	citation: represents the title of a work	global attributes**
<code>	computer code text: represents a fragment of computer code. This could be an XML element name, a filename, a command program, or any other string that a computer would recognize.	global attributes**
<col>	column: defines the attribute values for one or more columns in a table. Used inside of a table or colgroup	span: number of columns the tag should span
<colgroup>	column group: a group of one or more columns in the table that is its parent, if it has a parent and that is a table element	span: number of columns the tag should span
<command>	command button: a command that the user can invoke (like radio button or checkbox)	type: Specifies the type of command [checkbox command radio] label: gives the name of the command, as shown to the user icon: a URL to a picture that represents the command disabled: prevents the command from being executed [disabled] checked: Determines if the command is checked by default [checked] radiogroup: gives the name of the group of commands that will be toggled when the command is toggled title: gives a hint describing the command, which might be shown to the user to help them
<datagrid>	datagrid element: an interactive representation of tree, list,	disabled: defines whether the list is selectable [disabled]
<datalist>	dropdown list: a set of option elements that represent predefined options for other controls	global attributes**
<dd>	definition description: description, definition, or value, part of a term-description group in a description list (dl element), and the discourse, or quote, part in a conversation (dialog element)	global attributes**
	deleted text: represents a removal from the document	cite: a URL used to specify the address of a document that explains the change datetime: used to specify the time and date of the change
<details>	details element: represents additional information or controls which the user can obtain on demand	open: indicates whether the details are to be shown to the user [open]
<dialog>	dialog element: represents a document transcript, a dialog in a screenplay, an instant message log, or some other construct in which different players take turns	global attributes**
<div>	document block: creates a block level element with no special meaning	global attributes**

HTML 5 - extended

Tag	Info	Attributes	Tag	Info	Attributes
< dfn >	definition term: the defining instance of a term	title: the exact value of the term being defined	< input >	input field: a typed data field, usually with a form control to allow the user to edit the data	Attributes are dependant upon input type
< dl >	definition list: an association list consisting of zero or more name-value groups (a description list). Each group must consist of one or more names (dt elements) followed by one or more values (dd elements)	global attributes**		accept: specified to provide user agents with a hint of what file types the server will be able to accept alt: provides the textual label for the alternative button for users and user agents who cannot use the image autocomplete: determines if the data is considered sensitive and if autocomplete will be used [on off default] autofocus: determines if the input will get focus when a page loads [autofocus] checked: determines if the input will be checked by default [checked] disabled: prevents the input from being pressed [disabled] form: used to explicitly associate the button element with its form owner formaction: URL that specifies a form processing agent formenctype: specifies the content type used to submit the form to the server [application/x-www-form-urlencoded] multipart/form-data text/plain] formmethod: which HTTP method will be used to submit the forms data [get post put delete] formvalidate: indicate whether the form is to be validated during submission [formvalidate] formtarget: specifies the target where the form is submitted [blank _parent _self _top] height: height of the input in pixels list: used to identify an element that lists predefined options suggested to the user max and max: indicate the allowed range of values for the element maxlength: controls the maxlen length of the input to a control multiple: indicates whether the user is to be allowed to specify more than one value [multiple] name: elements name pattern: specifies a regular expression against which the control's value is to be checked placeholder: a short hint intended to aid the user with data entry readonly: determines if the control is readonly [readonly] required: determines if the input is required before the form submits [required] size: gives the number of characters that, in a visual representation, the user agent is to allow the user to see while editing src: URL to an image (image button) step: indicates the granularity that is expected (and required) of the value type: controls the data type (and associated control) of the element [hidden text search tel url email password datetime date month week time datetime-local number range color checkbox radio file submit image reset button] value: sets the element's value width: width of the input in pixels	
< dt >	definition term: the term, or name, part of a term-definition pair, a description list (dl element), and the talker, or speaker, part of a talker-discourse pair in a conversation (dialog element)	global attributes**	< ins >	inserted text: an addition to the document	cite: a URL used to specify the address of a document that explains the change datetime: used to specify the time and date of the change
< em >	emphasized text: represents stress emphasis of its contents.	global attributes**	< kbd >	keyboard text: user input (typically keyboard input, although it may also be used to represent other input, such as voice commands)	global attributes**
< embed >	embed element: an external (typically non-HTML) application or interactive content	src: URL of the resource being embedded type: gives the MIME type of the plugin to instantiate height: height of the embedded content in pixels width: width of the embedded content in pixels	< label >	label: caption in a user interface	for: specified to indicate a form control with which the caption is to be associated
< fieldset >	fieldset element: a set of form controls grouped under a common name	disabled: controls whether all the form control descendants are disabled [disabled] form: used to explicitly associate the fieldset element with its form owner name: gives the name of the form control	< legend >	fieldset title: sets the title of a fieldset element	global attributes**
< figure >	figure element: some flow content, optionally with a caption, that is self-contained and is typically referenced as a single unit from the main flow of the document	global attributes**	< li >	list item: represents a list item of an Ordered (OL) or Unordered list (UL)	value: used in an Ordered List (OL) to set the display value
< footer >	footer element: represents a footer for the section it applies to	global attributes**	< link >	resource link: allows authors to link their document to other resources	href: destination resource of the hyperlink rel: relationship between the document containing the hyperlink and the destination resource [alternate archives author bookmark controls external first last icon index icon last icon next last referrer previous pingback prefetch prev search stylesheet sidebar tag up] media: describes for which media the target document was designed hreflang: gives the language of the linked resource type: gives the MIME type of the linked resource sizes: gives the sizes of icons for visual media
< form >	form element: represents a collection of form-associated elements, some of which can represent editable values that can be submitted to a server for processing	accept-charset: gives the character encodings that are to be used for the submission action: URL that specifies a form processing agent autocomplete: determines if form elements will have their autocomplete turned on or off by default [on off] enctype: specifies the content type used to submit the form to the server [application/x-www-form-urlencoded] multipart/form-data text/plain] method: which HTTP method will be used to submit the forms data [get post put delete] name: elements name novalidate: indicate whether the form is to be validated during submission [novalidate] target: gives the target when the form is submitted [_blank _parent _self _top]	< mark >	marked text: a run of text in one document marked or highlighted for reference purposes, due to its relevance in another context	global attributes**
< h1h6 >	headers (1-6): represent headings for their sections. elements have a rank given by the number in their name	global attributes**	< map >	image map: in conjunction with any area element descendants, defines an image map	name: gives the map a name so that it can be referenced
< head >	head element: contains information about the document	none	< menu >	menu list: a list of commands	label: sets a visible label for the menu type: indicates the kind of menu being declared [context toolbar list]
< header >	header element: represents a group of introductory or navigational aids	global attributes**			
< hgroup >	heading group: used to group a set of h1-h6 elements when the heading has multiple levels, such as subheadings, alternative titles, or taglines	global attributes**			
< hr >	horizontal rule: creates a horizontal rule (line)	global attributes**			
< html >	HTML document: root of an HTML document	manifest: a URL to the address of the document's application cache manifest			
< i >	italic text: indicates the text is to be rendered with emphasis	global attributes**			
< iframe >	inline frame: represents a nested browsing window	src: URL of a page that the nested browsing context is to contain name: elements name sandbox: enables a set of extra restrictions on any content hosted by the iframe [allow-same-origin allow-forms allow-scripts] seamless: indicates whether the iframe element's browsing context is to be rendered in a manner that makes it appear to be part of the containing document [seamless] height: height of the frame in pixels width: width of the frame in pixels			
< img >	image: represents an image	alt: text to display if the image can not src: URL to the image file usemap: name of the map to use for the image ismap: provides access to a server-side image map height: height of the image in pixels width: width of the image in pixels			

HTML 5 - extended

Tag	Info	Attributes	Tag	Info	Attributes
<meta>	meta information: sets meta information for the page (like title, description)	charset: specifies the character encoding used by the document content: sets the value of the document metadata http-equiv: sets a pragma directive [content-language content-type default-style refresh] name: set the name of the meta information	<select>	selectable list: a control for selecting amongst a set of options	autofocus: determines if the controls gets focus when the page loads [autofocus] disabled: prevent the selection of an item [disabled] form: form to associate the select with multiple: allows the selection of multiple items [multiple] size: gives the number of options to show to the user
<meter>	meter element: scalar measurement within a known range, or a fractional value	high: specifies the range that is considered to be the "high" part low: specifies the range that is considered to be the "low" part min: specifies the lower boundary max: specifies the upper boundary optimum: specifies the range that is considered to be the "optimum" part value: current location within the range	<small>	small text: small print or other side comments	global attributes**
<nav>	navigation element: section of a page that links to other pages or to parts within the page: a section with navigation links	global attributes**	<source>	source element: allows authors to specify multiple media resources for media elements.	media: gives the intended media type of the media resource src: URL of the media resource type: gives the MIME type of the source
<noscript>	noscript section: represents nothing if scripting is enabled, and represents its children if scripting is disabled	global attributes**		span: used for an inline element	global attributes**
<object>	embedded object: an external resource, which, depending on the type of the resource, will either be treated as an image, as a nested browsing context, or as an external resource to be processed by a plugin	data: specifies the address of the resource name: valid browsing context name usemap: name of the map to use for the image form: form to associate the object with type: gives the MIME type of the plugin to instantiate height: height of the embedded content, in pixels width: width of the embedded content in pixels		strong: represents strong importance for its contents	global attributes**
	ordered list: list of items, where the items have been intentionally ordered	start: the ordinal value of the first list item reversed: indicates that the list is a descending list [reversed]	<style>	style definition: allows authors to embed style information in their documents	media: says which media the styles apply to type: gives the MIME type (default: text/css) scoped: indicates that the styles are intended just for the subtree rooted at the style element's parent element [scoped]
<optgroup>	option group: a group of option elements with a common label	disabled: disables all options in the group [disabled] label: gives the name of the group, as shown to the user	<sub>	script: subscript text	global attributes**
<option>	option element: an option in a select element or as part of a list of suggestions in a datalist element	disabled: prevent any clicks on an option item [disabled] label: provides a label for element selected: determines if the option is selected by default [selected] value: provides a value for element	<sup>	superscript: superscript text	global attributes**
<output>	output element: the result of a calculation	form: used to explicitly associate the output element with its form owner for: allows an explicit relationship to be made between the result of a calculation and the elements that represent the values that went into the calculation or that influenced the calculation	<table>	table element: represents data with more than one dimension, in the form of a table	global attributes**
<p>	paragraph: creates a paragraph	global attributes**	<tbody>	table body: represents a block of rows that consist of a body of data for a table	global attributes**
<param>	parameter element: defines parameters for plugins invoked by object elements. It does not represent anything on its own	name: gives the name of the parameter. value: gives the value of the parameter.	<td>	table cell: represents a data cell in a table	colspan: sets how many columns a cell will span headers: space separated list of ids corresponding to the th ids and give header information for the cell
<pre>	preformatted text: represents a block of preformatted text	global attributes**	<textarea>	text area: a multiline plain text edit control for the element's raw value	autofocus: determines if the textarea gets focus when the page loads [autofocus] cols: specifies the expected maximum number of characters per line disabled: prevents entry of text [disabled] form: form to associate the textarea with readonly: control whether the text can be edited by the user or not [readonly] required: be required to enter a value before submitting the form [required] rows: specifies the number of lines to show maxlength: controls the maximum amount of characters which can be entered placeholder: a hint intended to aid the user with data entry wrap: defines how text is wrapped [soft hard]
<progress>	progress element: represents the completion progress of a task.	max: specifies how much work the task requires in total value: specifies how much of the task has been completed	<tfoot>	table footer: the block of rows that consist of the column summaries (footers) for a table	global attributes**
<q>	short quotation: phrasing content quoted from another source	cite: a URL of a page where the quote was taken from	<th>	table header: represents a header cell in a table	colspan: determines how many columns a cell will span rowspan: determines how many rows a cell will span headers: space separated list of ids corresponding to the th ids and give header information for the cell scope: determines where the cell provides its header information [col colgroup row rowgroup]
<ruby>	ruby annotations: allows one or more spans of phrasing content to be marked with ruby annotations	global attributes**	<thead>	table header: the block of rows that consist of the column labels (headers) for a table	global attributes**
<rp>	ruby text parentheses: can be used to provide parentheses around a ruby text component of a ruby annotation	global attributes**	<time>	date/time: a precise date and/or a time in the Gregorian calendar	datetime: date/time using the Gregorian calendar
<rt>	ruby text component: marks the ruby text component of a ruby annotation	global attributes**	<title>	title element: sets the title of the document	none
<samp>	sample: sample output from a program or computing system.	global attributes**	<tr>	table row: a row of cells in a table	global attributes**
<script>	script element: allows authors to include dynamic script and data blocks in their documents	async: the script will be executed asynchronously, as soon as it is available [async] type: gives the MIME type of the script or format of the data defer: the script is executed when the page has finished parsing [defer] src: gives the address of the external script resource to use charset: specifies the character encoding of the external script resource		unordered list: a list of items, where the order of the items is not important	global attributes**
<section>	section element: represents a generic document or application section	cite: a URL of a page where the section was taken from	<var>	variable: this could be an actual variable in a mathematical expression or programming context	global attributes**
			<video>	video element: a video or movie	poster: URL of an image file that the user agent can show while no video data is available autoplay: determine if the audio will be buffered [autoplay] autoplay: determine if the audio will automatically play [autoplay] controls: indicates that the author has not provided a scripted controller and would like the user agent to provide its own set of controls [controls] loop: sets whether the audio will start once the end is reached [loop] src: URL of the audio to play width: width of the video in pixels height: height of the video in pixels

attribute_value is B over the accepted value

3- Diseño y Desarrollo

Para realizar este proyecto, he realizado una serie de clases, en las cuales se divide el código, que juntando todas las partes, forma el proyecto completo.

He ido desarrollando muchas pruebas antes de conseguir la versión final, por lo cual hay algún fichero que se utilizaba para las primeras partes, pero que luego, en la versión final, han sido desestimadas, como es el caso de “menu.js” y “cuadrado.js”.

Así pues, comencemos a explicar todo el diseño que he realizado.

Antes de comenzar a explicar el diseño hay una cosa interesante del lenguaje javascript que se debe saber y es que no soporta directamente matrices de dos dimensiones.

Así como en Java o C++ definimos matrices de dos dimensiones “matriz[5][5]” (5x5 = 25 elementos), en Javascript se puede “simular” esta matriz de la siguiente forma:

```
var Matriz2D=new Array(5);  
for(i=0;i<=4;i++)  
{  
    Matriz2D[i]=new Array(5);  
}
```

Así hemos creado una matriz de 5 elementos, de los cuales cada elemento es una matriz. El resultado es el deseado: una matriz de dos dimensiones.

Para acceder a los elementos de la matriz multidimensional, se realiza como en los otros lenguajes:

Matriz2D[0][0] = primer elemento de la matriz.

Ahora si, comenzamos a explicar el diseño del proyecto:

3.1 - Clases, estructuras y ficheros

3.1.1- Estructura Inicio

Inicio
+canvas = null
+canvas2 = null
+width: int = 0
+width2: int = 0
+height: int = 0
+height2: int = 0
+ctx = null
+ctx2 = null
+init()
+init2()

La estructura Inicio es la que contiene la información necesaria de los 2 canvas que se utilizan a lo largo del proyecto. Tenemos las variables canvas y canvas2, las cuales guardan el elemento canvas, la altura y anchura de ambos y también el contexto correspondiente. Y tiene 2 funciones, init e init2, las cuales son las que cargan los valores anteriormente mencionados con los valores correspondientes a cada uno de los canvas.

3.1.2- Clase Square

Square(color, posx, posy)	
+x: int	= posx
+y: int	= posy
+color	= color
+longitud: int	= 10
+altitud: int	= 10
+setposition(posx:int, posy:int)	
+setposx(posx:int)	
+setposy(posy:int)	
+changecol(color1)	
+draw(ctx)	
+drawImg(img)	

Es lo que se llama objeto dibujable, el cual lo único que necesita saber es cuando y donde debe dibujarse, no necesita tener conocimiento de ninguna cosa.

Inicialmente, esta clase era la que mostraba los personajes (obviamente eran cuadrados, no tenían las imagen de personaje). Tenía las variables x e y para saber en que posición (x,y) era la inicial para dibujarse, el color que tenía cada cuadrado y la longitud y altitud para saber el tamaño del cuadrado.

Con tal de interactuar con estas variables y atributos, la clase Square dispone de una serie de funciones, como son:

- setPosition: recibe por parámetro 2 variables, que actualizan la posición inicial del cuadrado.
- setposx: recibe un parámetro, el cual actualiza la posición X del cuadrado.
- setposy: recibe un parámetro, el cual actualiza la posición Y del cuadrado.
- changecol: recibe un parámetro, el cual cambia el color del cuadrado.
- draw: recibe un parámetro, el contexto del canvas del HTML5, para saber en que lugar debe dibujar el cuadrado.
- drawImg: recibe por parámetro una imagen, la cual llamas para ser pintada en el momento concreto.

3.1.3- Clase *ImageData*

ImageData(id,context,src)
+context = context
+id = 'image'+id
+posX: int = 0
+posY: int = 0
+depth: int = 0
+img: Image = new Image()
+setPosition(posX:int,posY:int)
+setSize(width:int,height:int)
+draw()
+draw2(x:int,y:int)

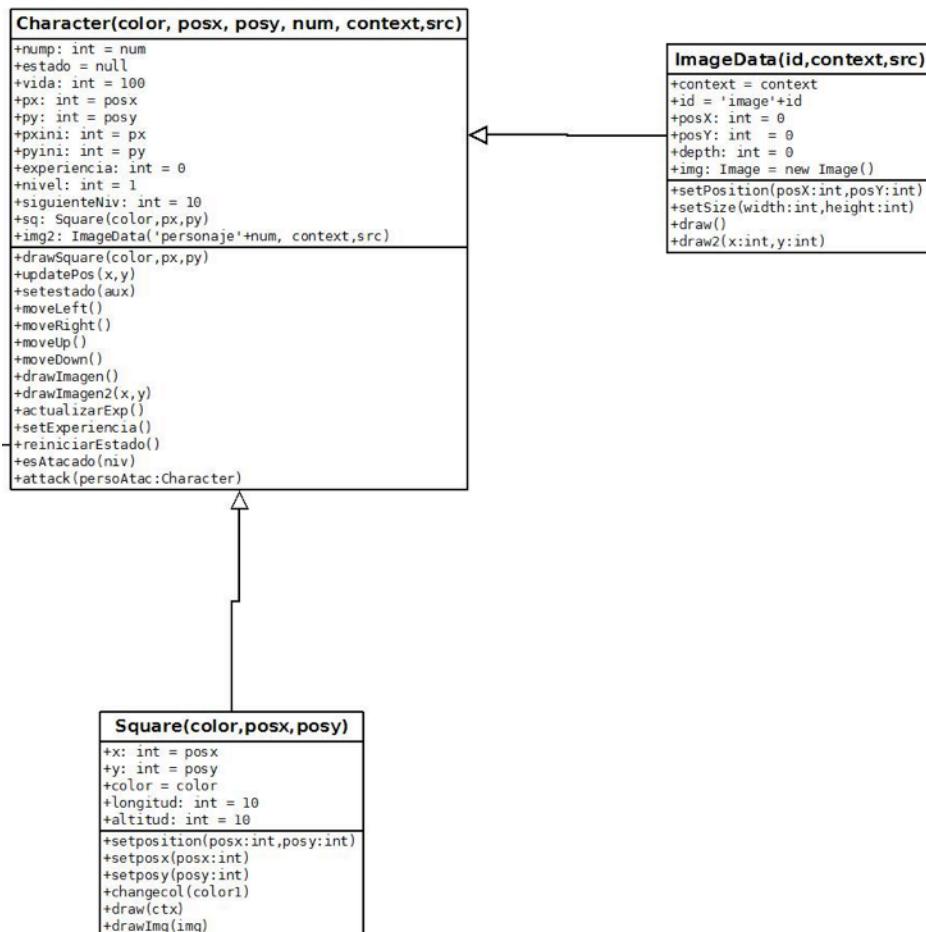
Esta clase ha sido creada para tratar con las imágenes, así como la clase ImageSet, que también trabaja con las imágenes, aunque de diferente manera, como ya se verá más adelante.

Esta clase recibe por parámetro el identificador de la imagen, el contexto en el cual será dibujada y la ruta de la imagen. Estos tres parámetros se almacenan en 3 variables. Pero para almacenar una de ellas, la ruta, primero se tiene que crear una instancia a la clase Image.

Como funciones tiene:

- setPosition: recibe por parámetro la posición a la cual debe actualizarse (x,y).
- setSize: recibe por parámetro la anchura y altura que tendrá la imagen.
- draw: Pinta la imagen en las posiciones marcadas por las 2 funciones anteriores.
- draw2: Esta función se ha creado para poder pintar la imagen del jugador activo en el segundo canvas.

3.1.4- Clase Character



Esta clase está un nivel por encima de la clase Square. No es un objeto dibujable, sino un objeto lógico.

Éste tiene conocimiento de las cosas propias de los personajes, así como la vida, la experiencia, el nivel del personaje,... , pero también tiene las funciones que permiten mover al personaje y atacar, como “moveleft” o “attak”.

La clase contiene diferente tipo de variables y funciones.

Primero de todo tiene un número identificativo, para saber en todo momento que personaje es. También tiene el “estado”, el cual identificará si es el estado activo (el que tiene que moverse en ese momento, “1”), es estado inactivo (está en pausa, hasta que le llegue el turno “0”) o está muerto (“2”).

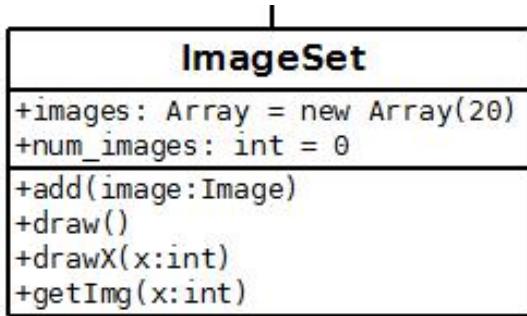
Tiene la variable vida, la cual le dice la cantidad de vida que tiene, también las variables de posición inicial del jugador (px,py), las cuales se irán modificando a medida que se mueva el personaje y también unas variables auxiliares (pxini, pyini), las cuales guardan la posición inicial del personaje, para que en el caso de que muera, pueda reaparecer en la posición inicial.

Así también tiene las variables experiencia de personaje, el nivel, la experiencia que le falta para subir de nivel, una variable que almacena un cuadrado (tal como se puede apreciar en el diagrama) y una variable que almacena una imagen.

La clase posee las siguientes funciones:

- drawSquare: Recibe el contexto en el cual ha de ser pintado. Sólo se pintará en el caso de que el estado del personaje no sea 2, es decir, que el personaje haya muerto.
- updatepos: Recibe un par de valores (x,y), los cuales sirven para actualizar la posición del personaje, de ahí que se actualice la posición de la imagen.
- setestado: Recibe el valor por el cual ha de ser sustituido.
- moveLeft: Actualiza la posición X del personaje. Se moverá tantas posiciones a la izquierda como niveles tenga. Acto seguido se actualiza la posición de la imagen asociada al personaje.
- moveRight: Es la inversa del moveLeft. Mueve tantas posiciones a la derecha como niveles tenga y también actualiza la imagen.
- moveUp: Actualiza la posición Y del personaje. Decrementará la “py” tantas posiciones como niveles tenga.
- moveDown: Inversa de moveUp. Incrementa “py” tantas posiciones como niveles tenga.
- drawImage: En el caso de que el personaje no esté muerto, dibuja la imagen asociada a él.
- drawImagen2: Lo que realiza es lo mismo que drawImage, lo que pasa es que se realiza la función para poder pintar la imagen en el canvas de información.
- actualizarExp: Esta función es la que incrementa la experiencia a conseguir por el personaje cada vez que sube un nivel.
- setExperiencia: Esta función incrementa la experiencia del personaje y, si la experiencia que tiene es mayor que la que necesita para subir de nivel, el nivel se incrementa, así como la vida, para posteriormente llamar a la función actualizarExp.
- reiniciarEstado: Esta función se requiere cuando un personaje muere. Se reinicia con los valores que tenía por defecto al empezar el juego.
- esAtacado: Esta función es requerida cuando un personaje es atacado por otro. Recibe por parámetro el nivel del jugador atacante, ya que cuanto mayor sea el nivel del atacante, mayor será el daño recibido. Si la vida que tiene el jugador es menor o igual que 10, morirá y se llamará a la función reiniciarEstado, para que el jugador pueda ser “resucitado” en el lugar inicial.
- attack: La función recibe por parámetro el personaje que sufre el ataque, para que así pueda “ser atacado”. Tras atacar al otro jugador, el atacante recibe experiencia por haber efectuado el ataque.

3.1.5- Clase ImageSet

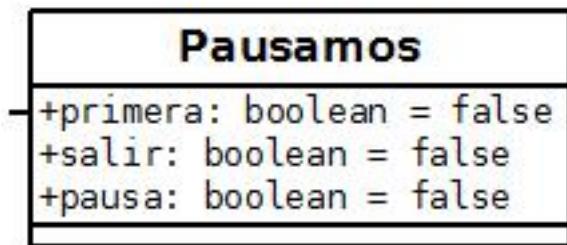


Esta clase, como se ha dicho antes, trata con imágenes, al igual que la clase `ImageData`, pero a un nivel diferente, ya que mientras `ImageData` sólo almacena datos de una sola imagen, `ImageSet` contiene un array de 20 posiciones, en el cual caben hasta 20 imágenes, las cuales se irán contabilizando con la variable `num_images`.

Como funciones tiene:

- `add`: Función que recibe una imagen por parámetro y la agrega al vector, y una vez hecho, incremente el numero de imágenes.
- `draw`: Función que se encarga de pintar todo el vector de imágenes en las posiciones correspondientes.
- `drawX`: Recibe por parámetro un valor, el cual se corresponde con la posición de la imagen que se quiere dibujar.
- `getImg`: Al igual que `drawX` recibe un valor correspondiente a la imagen que se quiere conseguir.

3.1.6- Estructura Pausamos



Esta estructura es una estructura “adjunta” en el fichero estadoPantGame.js, ya que sólo la clase pantGame accede a ella.

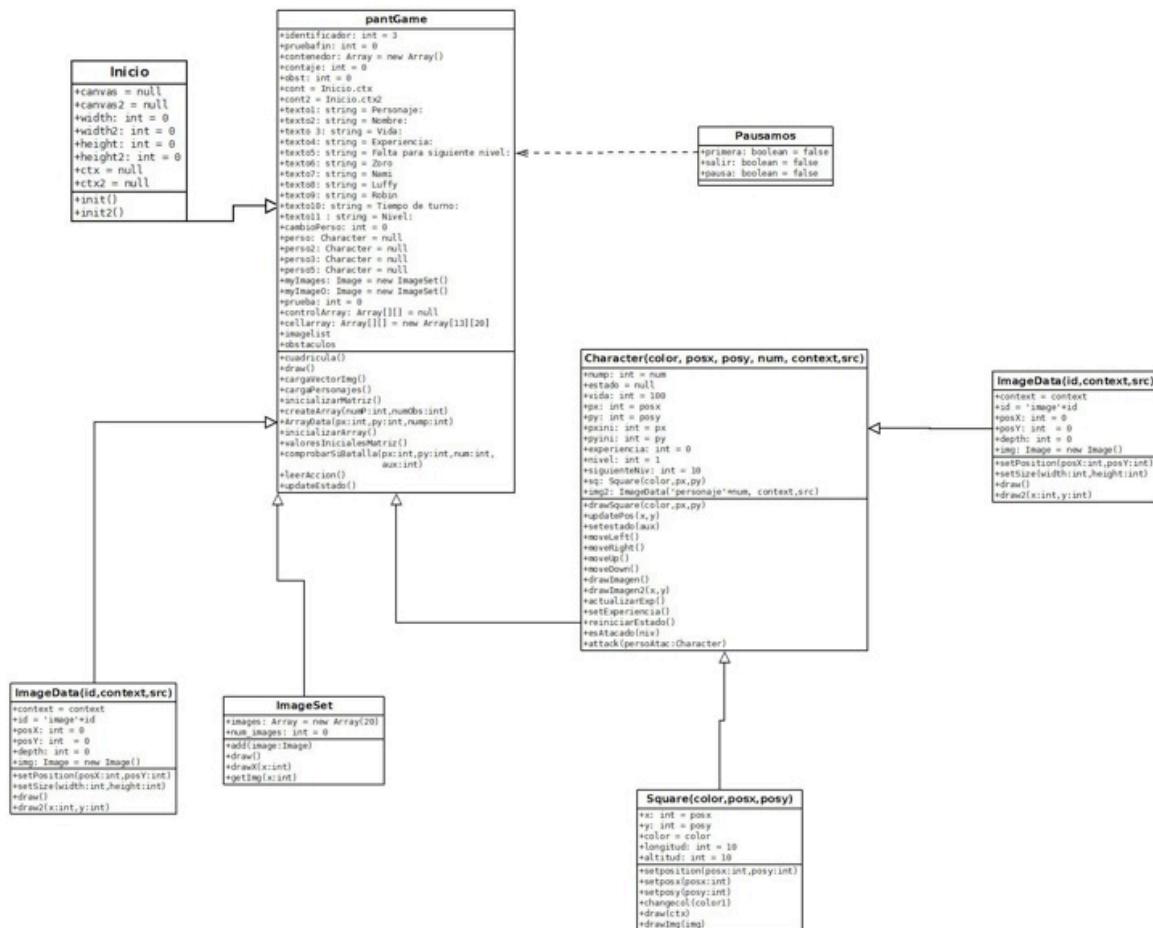
Así pues, tiene 3 atributos:

- primera: Variable empleada en el fichero “central.js”, para saber la primera vez que se entra en el estado (clase) pantGame, para que los sonidos colocados para que suenen durante el juego comiencen a sonar sólo la primera de las veces que se entra, ya que sino, como el bucle se repite, comenzaría a sonar indefinidamente y no es lo que se quiere.

- salir: Variable que indica que el juego se ha terminado debido a que los jugadores han decidido descansar. Cuando se decide salir del juego, la variable anteriormente descrita (primera) cambiará al valor inicial para que cuando se vuelva a empezar el juego, la música comience a sonar de nuevo.

- pausa: Variable que indica que el juego es pausado por alguno de los jugadores, pero no se termina la partida, sino que se reanudará más adelante, por eso la música que suena no termina, sino que se pausa también.

3.1.7- Clase pantGame



Como se puede apreciar, esta clase es la clase sobre la que se gestionan los personajes y las imágenes, aunque la clase Character también gestione imágenes, todo indica que va a parar a la clase pantGame.

Tiene un número identificativo, el cual tiene relevancia en otra clase. Tiene también diversas variables, como pruebefin y obst, que tienen importancia en las funciones de la clase.

Se crea un vector el cual contendrá a todos los personajes, para poder trabajar cómodamente con los personajes sin tener que llamarlos por su nombre. También tenemos unos cuantos strings, los cuales se utilizan el en segundo canvas.

Después se crean los personajes, así como los vectores de imágenes para guardar las imágenes contenidas en las variables “imagelist” y “obstaculos”.

Después se crea la variable que será un vector que contendrá las posiciones de los personajes (en una función se inicializará) y también se crea lo que es la matriz de juego (la cual se aprecia porque se ha dibujado también en el tablero de juego).

Durante la inicialización del juego (cuando se abre la pagina web), hay que tener en cuenta que previo a jugar, cuando se carga todo, ya se cargan los vectores de imágenes, se crean los personajes y se inicializan la matriz, el vector y se guardan los datos iniciales de los personajes en la matriz.

Explicado este detalle, procedamos a desarrollar las funciones de la clase:

- cuadricula: Esta función pinta la cuadrícula que se puede observar cuando se comienza a jugar. Primero hemos creado las lineas horizontales y después las verticales. Como cada imagen de personaje ocupa 50 unidades, pintará una linea cada vez que el resto de la division de “i” entre 50 de como resultado 0.

- draw: La función pinta varias cosas, y es importante que se haga en este orden, ya que sino unas imágenes superponen a otras y el juego no sería como tiene que ser. Primero se carga el escenario de juego, posteriormente la cuadrícula y después los obstáculos y personajes.

- cargaVectorImg: En esta función nos encontramos con 2 bucles, los cuales cargan las imagenes que están en “imagelist” y “obstaculos”. Para cada imagen, se crea una instancia a ImageDaya, se le administra una posición y tamaño y se añade al vector “myimages / myimageO”.

- cargaPersonajes: Esta función crea los personajes de juego, que por defecto son 4, pero en el caso de que se quieran más jugadores, basta con añadir las 3-4 lineas necesarias y administrarle una nueva imagen, o una ya existente. Primero se hace la instancia a la clase Character. Despues se le administra un estado y se guarda el personaje en el vector que los almacena. Finalmente se incrementa una variable para contabilizar los personajes que hay.

- inicializarMatriz: Esta función inicializa la matriz con el valor nulo por defecto, que en este caso es 12345, ya que el valor 0 lo contiene como identificador el primer personaje.

- createArray: La función recibe por parámetro el número de personajes y de obstáculos. Así, se crea el array necesario, con las posiciones justas y necesarias.

- ArrayData: Recibe por parámetro las variables px, py y numP de un personaje / obstáculo, los cuales los guarda en unas variables propias de la función.

- inicializarArray: Almacena en cada una de las posiciones del vector un ArrayData, es decir, la posición (x,y) y el identificador de cada personaje / obstáculo.

- valoresInicialesMatriz: Esta función inicializa la matriz con los valores de los personajes y obstáculos en las posiciones que habrá. Como la matriz ha sido definida cada 50 unidades, para saber la posición en la que va el elemento, se divide la px y py entre 50, para así saber la posición en la que va y lo que se almacena en la posición es el identificador del personaje / obstáculo.

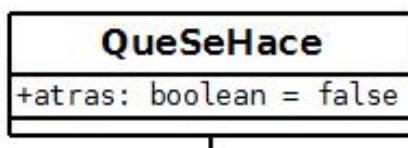
- comprobarSiBatalla: La función recibe por parámetro la posición (x,y) a la que se intenta acceder, el identificador del personaje que hace la llamada y la posición del vector de personajes que ocupa el personaje que hace la llamada. Primero se comprueba si la

posición a la que se accede está o no vacía. En el caso de que no lo esté, se mirará si es 100, 85 o 30 (los obstáculos) i en el caso de que no sea ninguno, se llamará a la función attack, donde se le pasará el personaje que ocupaba la posición, para que así pueda realizar el ataque correspondiente.

- leerAcción: En esta función se realizan varias cosas. Primero se mira si se ha pulsado la tecla que indica el final del juego. En el caso de que no esté pulsada, se mira que no se haya pulsado la tecla que indica que el juego se pausa. Una vez hechas las comprobaciones, se mira cual de los personajes es el que tiene el estado activo, para pasar a comprobar cual de las teclas ha sido pulsada. Si se ha caído sobre un obstáculo, volveremos atrás, para que no se “coma” el obstáculo, y éste pueda realizar su acción tantas veces como personajes choquen contra ellos. En el caso de que la vida del personaje sea inferior a 0, se reiniciará su estado y se añadirá en la posición correspondiente de la matriz. Finalmente la tecla que haya sido pulsada se pondrá con el valor por defecto y la variable que permite cambiar de jugador se incrementará (la que permite que cada jugador tenga 10 segundos para moverse). Para terminar, se mira si los 10 segundos han transcurrido o se ha pulsado la tecla intro (la que determina que el jugador decide pasar el turno). En esta última parte, se mirará si el jugador activo era el último del vector de personajes, en cuyo caso el siguiente jugador activo será el primero de la lista. En el caso de que se haya decidido pausar el juego, la variable pausa de la clase Pausamos cambiará para que el nuevo estado sea el de pausa.

- updateEstado: Lo primero que se realiza es una limpieza de pantalla de ambos canvas. Posteriormente se escriben algunos de los strings definidos, los que son iguales para todos los jugadores. Posteriormente, se observa cada jugador del vector de personajes y se mira cual es el activo, para pintar la información de cada uno, como es el identificador, el nombre del personaje, la vida, experiencia, lo que le falta para subir de nivel, el nivel y la imagen del personaje. Para terminar, se calcula el tiempo que ha transcurrido en segundos (para saber cuento tiempo le queda para realizar la acción) y se muestra en el segundo canvas.

3.1.8- Estructura QueSeHace



Esta estructura tan sólo tiene una variable, la cual se utiliza en la clase pantHelp, por tanto, la variable permitirá volver de la pantalla de ayuda a la pantalla de inicio sin ningún tipo de problema.

3.1.9- Estructura Selecciones

Selecciones	
+jugar:	boolean = false
+ayuda:	boolean = false
+salir:	boolean = false

La estructura selecciones contiene 3 variables, las cuales se reflejan en la primera pantalla que se muestra en el juego, ya que son las 3 opciones que se tienen.

- jugar: Hace que el estado de juego cambie de pantIni a pantGame.
- ayuda: Hace que el estado de juego cambie de pantIni a pantHelp.
- salir: Hace que el estado de juego cambie de pantIni a pantOut.

3.1.10- Estructura Vars

Vars	
+volver:	boolean = false

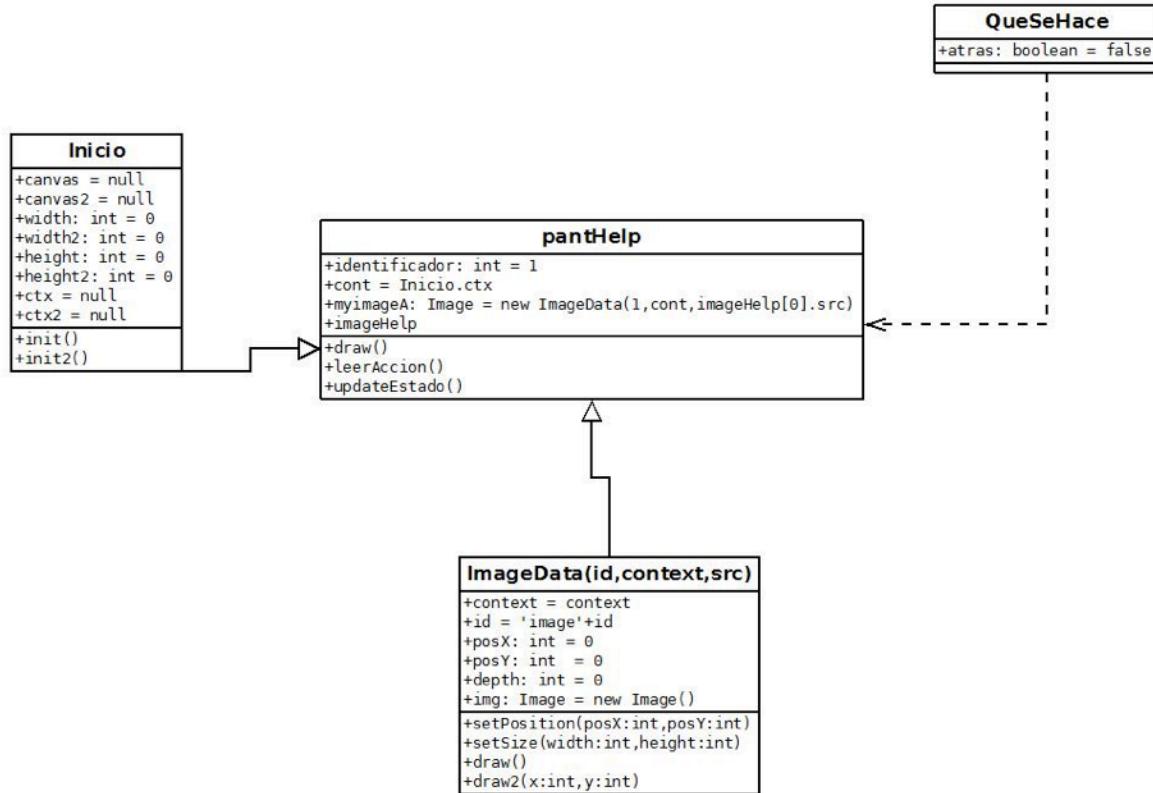
Esta estructura es requerida por una clase, la pantPause. Solamente tiene un atributo, volver, el cual hace que el estado de juego cambie de pantPause a pantGame.

3.1.11- Estructura volver

Volver	
+inicio:	boolean = false

Esta estructura tiene un único atributo, inicio, el cual se utiliza en la clase pantOut, que sirve para que si la clase pantOut accede a él, significa que el estado debe cambiar de pantOut a pantIni.

3.1.12- Clase pantHelp

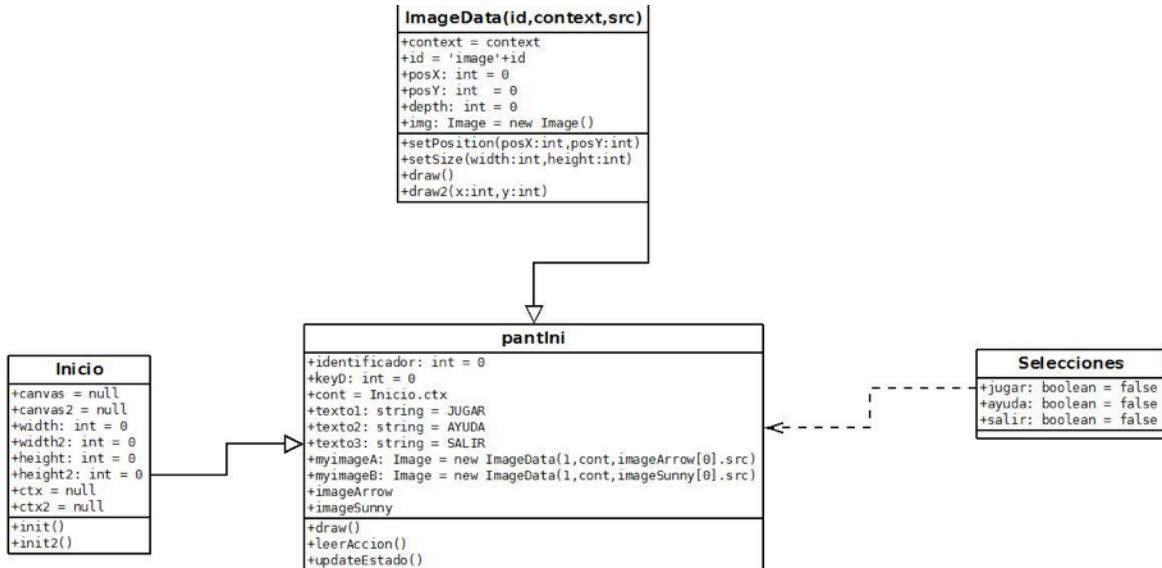


Esta clase, al igual que la `pantGame`, tiene una variable que identifica la clase. También almacena el contexto del canvas y crea una variable que tiene la imagen almacenada en la variable `imageHelp`.

Las funciones que tiene son las siguientes:

- `draw`: Esta función se encarga de pintar la imagen guardada en la variable `"myimageA"`.
- `leerAccion`: La única acción que se realiza desde esta clase es cuando se pulsa el botón izquierdo del ratón, lo que quiere decir que se quiere volver atrás, a la pantalla de inicio.
- `updateEstado`: Esta función limpia el canvas correspondiente.

3.1.13- Clase pantIni



Como el resto de las clases pant--- , tiene una variable que lo identifica. Tiene otra variable que se irá incrementando o decrementando según se pulse las teclas down y up. También se crean 2 variables que guardan las imágenes de “imageArrow” e “imageSunny”.

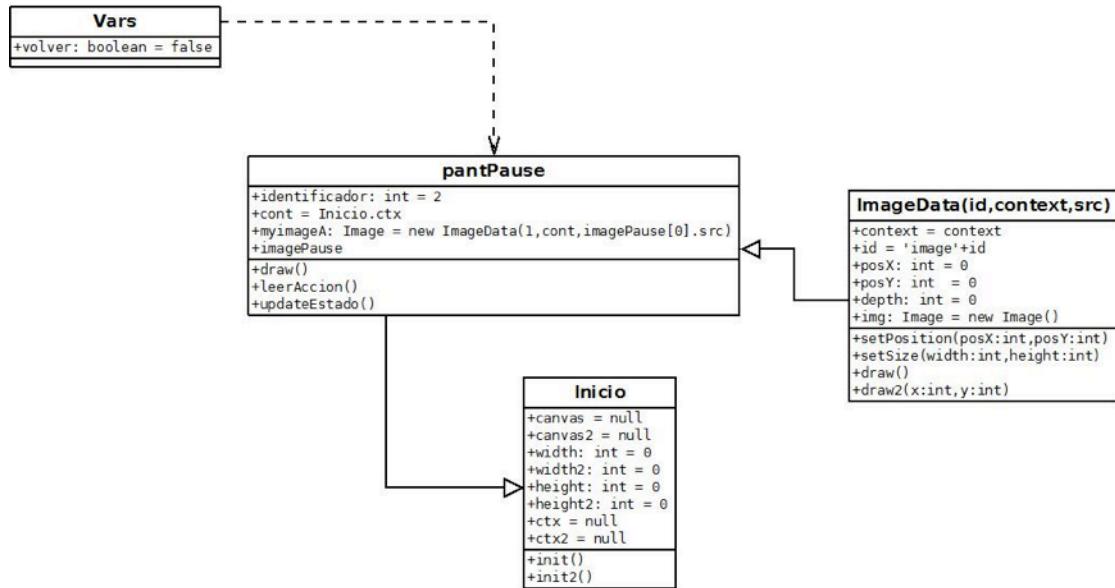
Las funciones que tiene son las siguientes:

- draw: Esta función pinta la variable que contiene “imageArrow”.

- leerAcción: Primero se mira si se ha pulsado la tecla down o la up, para incrementar o decrementar la variable. Si se llega al tope, se pone a 0 o a 2. En cuanto se ha mirado esto, se observa si se ha pulsado el botón izquierdo del ratón, el cual determina cual de los 3 atributos de la clase Selecciones se ha de activar.

- updateEstado: Lo primero que realiza esta función es limpiar la pantalla. Después pinta la variable “imageSunny”. A continuación escribe en pantalla 3 strings. Finalmente observa cual de los casos se cumple (referente a si se pulsa la tecla down o up) y se actualiza la posición e la variable “imageArrow”.

3.1.14- Clase pantPause

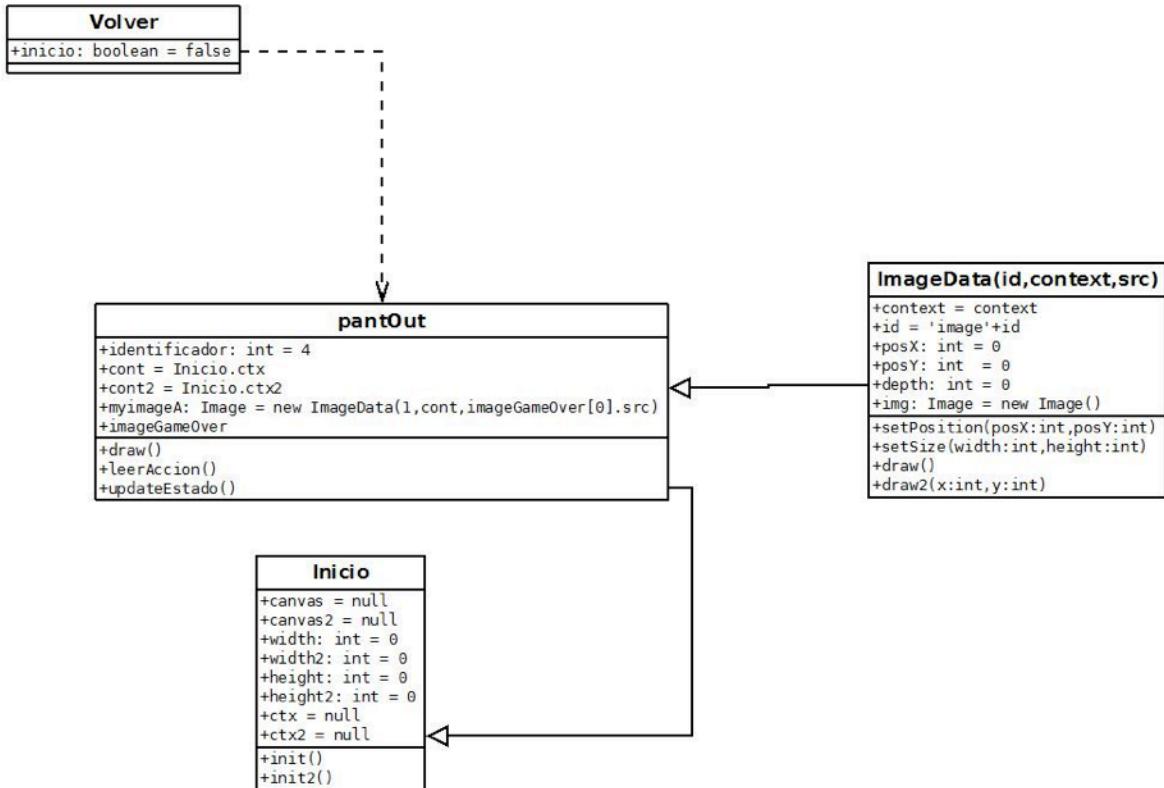


Esta clase también tiene el elemento identificador, así como la variable que guarda el contexto con el cual se está trabajando y una variable que guarda la imagen “imagePause”.

Las funciones que tiene la clase son las siguiente:

- `draw`: La función pinta la variable que contiene “imagePause”.
- `leerAcción`: Se observa si se ha pulsado la tecla “p”, que querrá decir que el juego ya no requiere estar pausado, con lo que la variable volver de la clase `Vars` se activará y realizará su función.
- `updateEstado`: Se limpia la pantalla.

3.1.15- Clase pantOut



Esta clase, al igual que el resto, pantIni, pantHelp, pantGame y pantPause, tiene un número que la identifica, el cual se utiliza en otra clase. Aparte de eso, se inicializan los canvas y contextos de la clase Inicio y se almacenan los contextos en 2 variables. Así mismo, también obtenemos en un atributo la imagen guardada en “imageGameOver”.

Las funciones que posee la clase son las siguientes:

- draw: La función dibuja la imagen guardada en la variable “imageGameOver”.
- leerAcción: En el caso de que se pulse el botón izquierdo del ratón, la variable inicio de la clase Volver se activará.
- updateEstado: Esta función limpia los 2 canvas, puesto que al haber cambiado a la clase pantOut, desde la pantalla pantGame tal vez no había dado tiempo de limpiar el canvas para dejarlo sin nada.

3.1.16- Estructura Tecles

Tecles
+keyup: boolean = false
+keydown: boolean = false
+keyright: boolean = false
+keyleft: boolean = false
+keyesc: boolean = false
+keypause: boolean = false
+keyintro: boolean = false
+keyhelp: boolean = false

Esta estructura interactúa con la el código HTML5 “document”, la cual activa o desactiva las variables de esta clase según convenga. Así pues tenemos:

- keyup: correspondiente a la “flecha arriba”.
- keydown: correspondiente a la “flecha abajo”.
- keyright: correspondiente a la “flecha derecha”.
- keyleft: correspondiente a la “flecha izquierda”.
- keyesc: correspondiente a la “tecla escape”.
- keypause: correspondiente a la “tecla p”.
- keyintro: correspondiente a la “tecla intro”.
- keyhelp: correspondiente a la “tecla h”. Aunque en principio esta variable era la que permitiría ir y salir del menú de ayuda, al final no se ha utilizado.

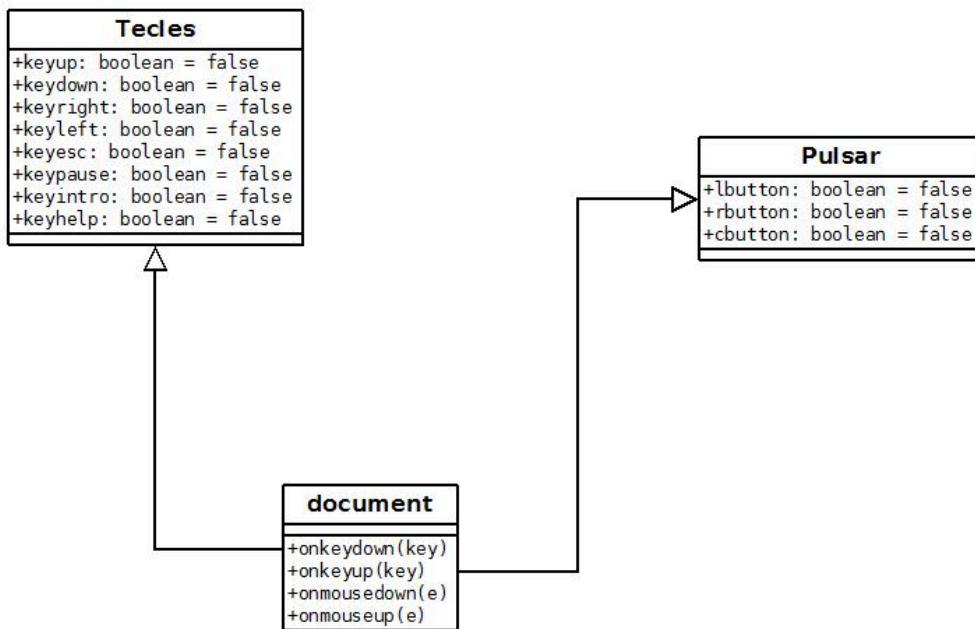
3.1.17- Estructura Pulsar

Pulsar
+lbutton: boolean = false
+rbutton: boolean = false
+cbutton: boolean = false

Esta estructura, al igual que la estructura Tecles, interactúa con el código HTML5 “document”. Esta tiene que ver con los eventos que se puedan producir con el ratón.

- lbutton: Esta variable se corresponde con el botón izquierdo del ratón.
- rbutton: Se corresponde con el botón derecho del ratón.
- cbutton: Se corresponde con el botón central del ratón.

3.1.18- Código HTML5 document



Este código es el que activa y desactiva las variables de las otras dos clases, de Tecles y de Pulsar. Por eso tiene las funciones necesarias para poder realizar las acciones que requieren:

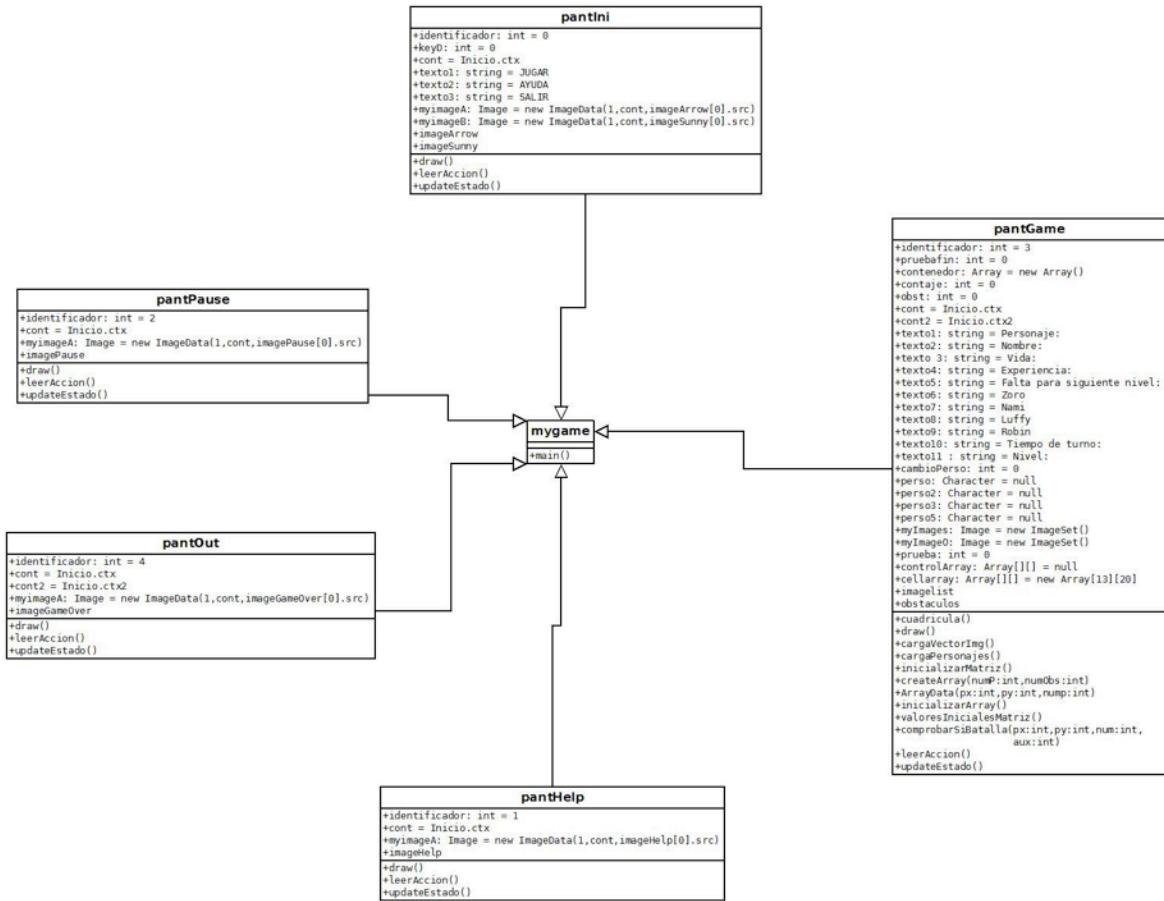
- `onkeydown`: Recibe por parámetro la señal de que una tecla ha sido pulsada. Cada tecla es representada por un número ASCII, el cual se guarda en una variable. Entonces se mira cual de las opciones, si se ha pulsado una de las teclas activas para el juego, ha sido marcada, para modificar la variable de la clase Tecles correspondiente. Lo que hace es activar las teclas.

- `onkeyup`: Igual que la anterior, recibe la señal de que una tecla ha dejado de ser pulsada. La función que realiza es la inversa de la anterior, es decir, si la anterior activaba las teclas, ésta las desactiva.

- `onmousedown`: Esta función, como las dos anteriores, recibe un dato por parámetro, el cual es el identificativo del botón del ratón que ha sido pulsado. El número 1 corresponde con el botón izquierdo del ratón, el 2 con el derecho y el 4 con el central.

- `onmouseup`: También recibe el numero identificativo del botón del ratón pulsado por parámetro. Mientras que la función anterior, `onmousedown`, activaba los botones del ratón, esta función lo que hace es desactivarlos.

3.1.19- Clase mygame



Como se puede apreciar en el diagrama, esta clase, **mygame**, es la encargada de gestionar todas las otras clases.

Lo primero que nos encontramos cuando comienza la clase, son 5 variables, correspondientes cada una a una instancia de las 5 clases que se muestran junto a la clase **mygame** en esta parte del diagrama (**pantIni**, **pangHelp**, **pantPause**, **pantGame**, **pantOut**).

Justo después tenemos una variable que guardará la instancia a la clase activa en cada instante. Como es normal, la primera clase activa es la **pantIni**, la clase de inicio donde se muestra lo que se quiere hacer, si jugar, ver la ayuda (que tipo de teclas y botones sirven) o salir.

Ahora lo que viene es el bucle principal de juego, el **main**. Este bucle , como se puede apreciar en la línea final del archivo, se ha hecho que sea un bucle que se repita 25 veces por segundo, para controlar cualquier evento que pueda suceder en todo momento y mostrarlo por pantalla.

El trabajo que tiene el main es diverso. Por una parte se tienen las funciones que actualizan el juego, **updateEstado**, **leerAccion** y **draw**, pero por otra tenemos las comprobaciones para saber en que estado de juego se está y si se necesita cambiar.

Ahora es cuando tienen relevancia los elementos identificadores de cada una de las otras 5 clases principales. Se mira cual de los identificadores es:

- 0 para **pantIni**.
- 1 para **pantHelp**.
- 2 para **pantPause**.
- 3 para **pantGame**.
- 4 para **pantOut**.

Una vez dentro del estado que sea, se miran las variables de las clases adjuntas para saber que hay que hacer a continuación.

- Selecciones para **pantIni**.
- QueSeHace para **pantHelp**.
- Vars para **pantPause**.
- Pausamos para **pantGame**.
- Volver para **pantOut**.

En el caso de que estemos en pantIni miraremos las variables de la clase **Selecciones**, para saber si tenemos que ir a la ayuda, salir del juego o ir a jugar.

En el caso de que sea pantHelp, miraremos si la clase **QueSeHace** determina que se debe volver a la pantalla inicial o no.

Si estamos en pantPause, se observará la clase **Vars**, para ver si el juego que había sido pausado ya no requiere que siga estandolo, en cuyo caso la música, que se había pausado al igual que el juego, volverá a sonar.

Si estamos en pantGame, miraremos cual de las variables de **Pausamos** requiere atención. Si la variable pausa se ha activado, pausaremos la música y cambiaremos al estado pantPause. Si por el contrario se ha activado la variable salir, pararemos la música, modificaremos la variable “primera” y reiniciaremos el juego para cuando se vuelva a comenzar a jugar. En el caso de que haya sido la variable primera la que requiere atención (se entrará cuando la variable sea false y no true), comenzará a sonar la música. Esto se ha hecho para que a cada vuelta de bucle, si no se cambia de estado, la música comenzaría a sonar una y otra vez.

Por último, si estamos en pantOut, miraremos la clase **Volver**, para saber si se requiere cambiar al estado inicial de nuevo o no.

3.1.20- Archivos obtenidos para poner sonidos al juego

Hemos obtenido la librería “SoundManager”, la cual nos permite interactuar con los sonidos.

El fichero que se encarga de crear y preparar los sonidos es “sonido.js”, el cual se podía poner como código del fichero correspondiente a la pagina web “prueba.html”, pero me ha parecido que quedaba mejor si se ponía así.

```
soundManager.debugMode = false;
soundManager.preferFlash = false; // use HTML5 audio for MP3/MP4, if available
soundManager.onready(function() {
    // SM2 has loaded - now you can create and play sounds!
    music = soundManager.createSound({
        id: 'music',
        url: 'landing_party.mp3'
        // onload: myOnloadHandler,
        // other options here..
    });

    music.setVolume(100);
    music.play({
        loops: 5
    });
    // music.pause();
});
```

Este es una de las dos cargas de sonidos que hay, puesto que se ha diferenciado una música mientras se juega y un sonido, el cual sonará cuando un personaje se choque contra un obstáculo o contra otro personaje.

3.1.21- Fichero menu.js

Aunque en la versión final no se utiliza, ha servido para versiones previas, ya que antes de tenerlo separado en pantIni, pantGame, pantPause,..., estaba todo junto en esta clase.

También, mientras iba haciendo las diferentes clases, esta clase seguía teniendo utilidad, ya que como todavía no estaba implementada la clase **mygames**, probaba los cambios que iba haciendo en esta clase, para ponerlo en la correspondiente en el caso de que fuese, si no, arreglaba el error para posteriormente ponerlo en la clase en la que tenía que ir.

3.1.22- Fichero prueba.html

Este es el fichero base, sin él no puede ir nada.

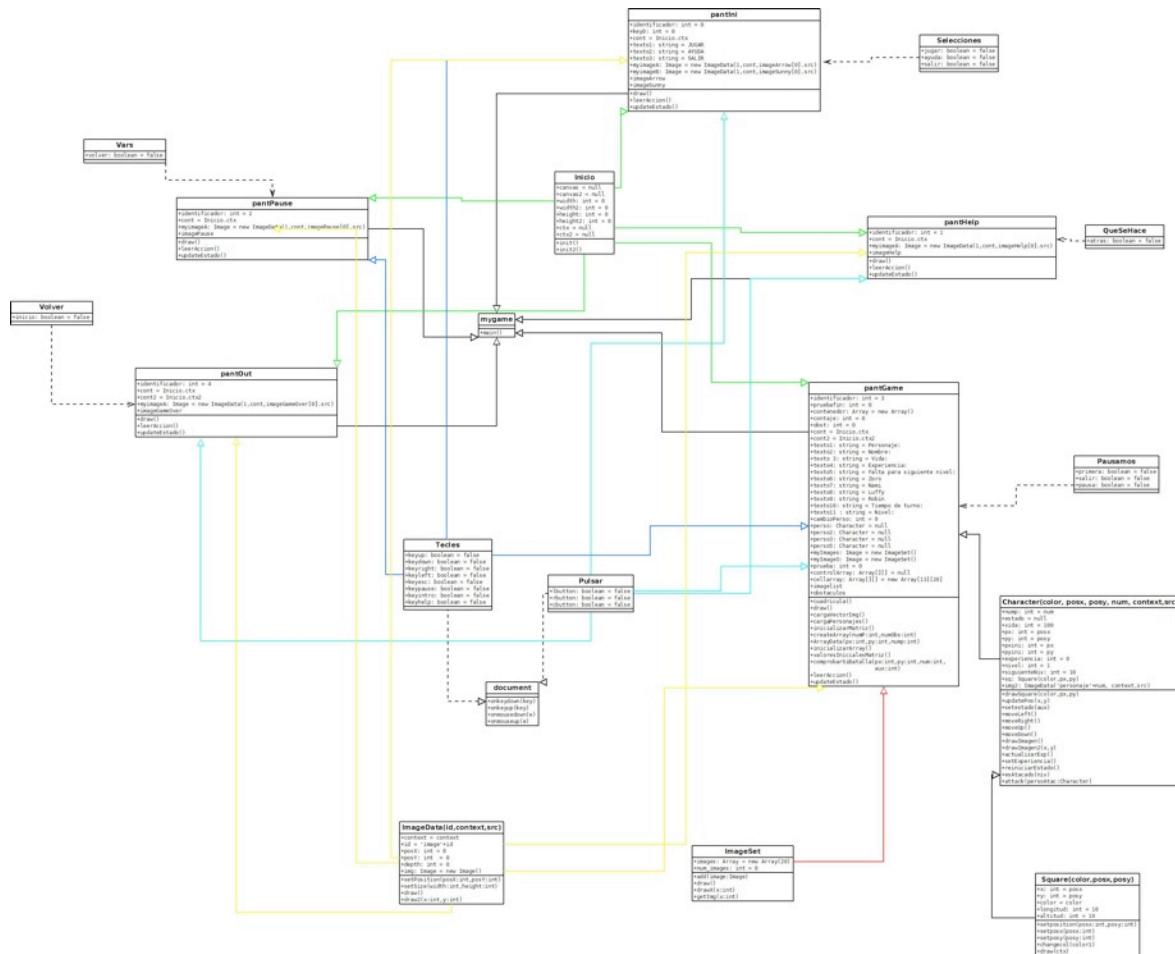
Lo primero que se le hace es ponerle el título y cargar todos los ficheros en el orden que tienen que ir, ya que sino se puede dar lugar a errores si se carga un fichero que necesita de otros antes que estos.

Así pues, primero cargamos la librería del soundmanager2, la que nos permite interactuar con sonidos, y acto seguido el fichero “sonido.js”, el cual activa y prepara los sonidos para cuando tengan que activarse.

Más tarde se van cargando el resto de los ficheros. Primero “librería.js”, puesto que es la que se encarga de los canvas y los contextos de éstos, después “image.js”, ya que todas las clases principales necesitan dibujar algún tipo de imagen. Despues se carga “personaje.js”, “cuadrado.js”, “teclado”, “raton”, “estadoPantIni”, “estadoPantHelp”, “estadoPantPause”, “estadoPantGame”, “estadoPantOut”, “menu.js” y por último, “central.js”, ya que éste necesita de todos los otros para funcionar.

Después se activan los sonidos con la etiqueta de HTML5 audio, la cual, junto con soundmanager2, permite que se puedan escuchar sonidos en el juego.

El body lo que hace es cargar la clase **mygame**, la cual es la central de todas, la que ha de funcionar, ya que el resto de clases, de una forma u otra, terminan yendo a la clase **mygame**. Por último, se cargan e identifican los 2 canvas, el primero es el de juego, y el segundo es el de información.



(Aquí se ve muy pequeño, en la memoria impresa se ve más grande, ya que se ha realizado una copia en DIN A3 y se ven mejor las relaciones y el contenido de cada clase)

4- Evaluaciones (Pruebas para comprobar cada parte)

Para comprobar que el proyecto fuese correctamente, he realizado las siguientes pruebas.

Primero de todo fragmenté el proyecto en diferentes partes, para ir probando una a una, y no llegar al final del proyecto sin saber si funcionaba todo correctamente o había algún error.

Tal como se va explicando de alguna forma en el diseño y desarrollo, partía de lo base, de que había un objeto dibujable (el cuadrado), el cual sólo tenía que tener noción de pintarse en pantalla en el momento en el que fuese requerido, es decir, no tenía que saber nada del teclado, ratón, ni variables más globales, las cuales eran propias de un nivel superior, el objeto lógico (personaje), el cual tenía noción de más cosas, pero sin saber nada del teclado ni ratón.

La primera prueba de todas realizara era comprobar que los personajes crearan correctamente el cuadrado correspondiente y lo pintasen en la posición que se le había indicado por pantalla.

Después de realizar este tipo de prueba, implementé lo referente al teclado, y comprobé que según pulsaba las teclas, los personajes se movían acorde a la tecla pulsada, eso si, primero se hacía todo a la vez, no cada vez que se pulsara se movía un personaje, sino que lo hacían los 4 al mismo tiempo, y se quedaban los cuadrados anteriores pintados en pantalla.

Aquí es donde pensé en implementar el lo que llamé entonces “redibujar escenario”, lo que en la versión final del juego ha sido “updateEstado”. Lo que primero se realizaba era limpiar el canvas, es decir, todo lo que hubiera pintado antes de que se llamara a la función desaparecía de la pantalla. Después de hacer la limpieza, se pintaban los personajes (cuadrados) en la posición que tenían almacenada en su clase. Más adelante, en la versión final del proyecto, tras limpiar la pantalla, se pinta el escenario, después la cuadricula de juego, los obstáculos y personajes (si se está en el estado de Juego), o las imágenes y textos correspondientes al estado que se encontrara en ese momento.

Hasta aquí, lo que era las inicializaciones de las variables y la creación de los personajes estaba todo seguido, sin estar separado por funciones. Entonces pensé en separarlo todo en funciones, así como se puede apreciar en el fichero “menu.js”, el cual era el fichero base hasta más adelante.

Después de comprobar estas cosas, implementé el evento del ratón, para lo cual realicé una prueba que era que cada vez que se pulsara el botón izquierdo del ratón uno de los cuatro personajes (cuadrados) se pintaría en una posición determinada.

Llegados a este punto, para tratar con los diferentes personajes del juego, hice un vector el cual almacenaría todos los personajes, cada uno en una posición diferente. Este dato se ha ido probando a partir de este punto, ya que así no hacía falta poner “this.perso, this.perso2,...”, ahora con poner “this.contenedor[x]” era suficiente.

El siguiente punto se ha visto modificado a lo largo del proyecto, como el “resibujarEscenario”, primero se llamaba “actualizaJugadores”, para terminar siendo “leerAccion”. También es aquí donde se hace que sólo se mueva un jugador por vez. Se recorre un bucle que mira cual es el personaje activo y entonces mueve su cuadrado.

Hasta este punto, lo que pasaba era que los personajes se movían libremente por pantalla, sin tener ningún tipo de problema si uno se pintaba encima del otro. Llegados a este punto, fue cuando se empezaron a poner las reglas en el juego, es decir, que si un jugador se pintaba encima de otro, es que lo atacaba y por tanto, le quitaba vida y se iba incrementando su experiencia (esto se realiza en alguna de las funciones de los personajes, teniendo como base la función “attack”).

Para comprobar tanto que se atacara correctamente y la experiencia fuese incrementada, lo que hacía era llamar a la función “alert(variable a comprobar);” para verificar que las funciones realizaban su tarea correctamente.

Una primera prueba de cambiar de un estado a otro, como de pantIni a pantGame (aunque todavía no estaban implementadas las clases como tales), era que se cargaba una pantalla previa donde no había nada y no se mostraba la pantalla de juego hasta que no se pulsara el botón izquierdo del ratón. Aquí es donde el ratón ya realizaba una tarea más parecida a la final que la otra nombrada anteriormente.

Para probar por primera vez el código de las imágenes, que funcionase correctamente como debería, me serví de lo que más tarde se convirtió en pantPause, es decir, que cuando se pausase el juego, cargase una imagen, que no era la imagen final, era tan sólo una de prueba.

Lo que aquí se vio es que como cuando se pulsa la tecla, al levantarla se desactivaba, para comprobar que funcionase lo probé con un par de variables auxiliares, las cuales mantenían la tecla “p” activa hasta que no se pulsase por segunda vez. Lo que pasaba entonces era que aunque el juego estuviese pausado, si se pulsaban las teclas de dirección, los personajes seguían moviéndose. Esto se arregla al separar todas las clases.

Una de las siguientes cosas en probar fue que cada personaje tuviese su imagen propia, lo cual, al cargarlas todas con “imageSet”, se cargaban en un vector y no se podían separar. Aquí es donde se crearon las funciones “drawX” y “getImg”, para obtener cada una de las imágenes del vector por separado. Después esto se ha visto modificado al crear

la imagen del personaje dentro del personaje, no pasándole la imagen, con lo que las funciones han caído en desuso.

Fue en este punto en el cual creé la matriz de imágenes, para poder tener una estructura capaz de guardar la posición del personaje, o al menos, algún dato identificativo de cada personaje y obstáculo, para así poder controlar de mejor manera cuando y donde se producía el enfrentamiento entre 2 personajes o el choque entre un personaje y un obstáculo. Como he señalado al inicio del Diseño y Desarrollo, aquí encontré el problema de que Javascript no soporta las matrices multidimensionales directamente, cosa que se puede simular de la manera realizada.

Para hacer las cosas en orden, primero se inicializaba la matriz con valores nulos, para posteriormente inicializarla con los valores por defecto de los personajes y obstáculos.

La matriz lo que hace es guardar el elemento identificativo de cada personaje y obstáculo, pero tenemos un array que guarda la posición (x,y) y el elemento identificativo del personaje y obstáculo. Y cada vez que se mueve el personaje cambia la posición del vector y de la matriz, donde se comprueba que no haya nada interponiéndose en su camino.

Cuando llegué a este punto, me di cuenta de un fallo que había cometido y era que cada vez que se entraba en la función actualizaJugadores (que finalmente es la función leerAccion de la clase pantGame), se cambiaba de jugador, por eso me pensaba que estaba bien. Pero como a cada vuelta del bucle se llamaba a esta función, el personaje que se movía no tenía por qué ser el personaje que tuviera el turno.

Debido a este problema, se creó una variable llamada "cambioPerso". Esto se controló de la siguiente manera: Ya que el bucle principal se repite 25 veces por segundo, para darle tiempo al jugador para moverse, le he administrado 10 segundos, durante los cuales puede decidir cual es el mejor movimiento, si quiere realizarlo. Esto se hizo mediante la siguiente fórmula:

$$25 \text{ veces / segundo} = X \text{ veces / 10 seg} \quad (1)$$

o lo que es lo mismo:

$$X = 25 \text{ veces * 10 segundos} = 250 \text{ interacciones de bucle} \quad (2)$$

De esta forma se tiene un tiempo para mover cada uno de los jugadores, y no se tiene el problema anteriormente mencionado.

En el inicializarArray, hasta este punto lo tenía como una matriz multidimensional donde en cada fila había un personaje/obstáculo, y en cada columna un elemento, px, py, identificador. Para realizarlo de forma más eficiente, lo que hice fue cambiar esa matriz multidimensional por un vector de una dimensión, el cual hacía una instancia a la función ArrayData, la cual guardaba los elementos px,py e identificador.

Fue en este punto donde se decidió separar lo que había sido la clase principal “menu.js” en las diferentes clases y ficheros, “pantGame”, “mygame”,... Aún así continuaba trabajando con el fichero “menu.js”, para ir comprobando cambios que se iban realizando para comprobar que funcionasen, como poner los bucles de las imágenes en la función “cargaVectorImg” o poner la creación de los personajes en la función “cargaPersonajes”.

Durante el transcurso de estas pruebas vi que había cometido un fallo, y es que no me había dado cuenta que las posiciones (x,y) de los personajes y las posiciones (x,y) de la matriz eran inversas, es decir, si en el personaje era (x,y) en la matriz era (y,x). Esto ocurrió debido a que los personajes, cuando se mueven a derecha/izquierda, se mueven horizontalmente, en el eje X, mientras que si lo hace de forma vertical estamos en el eje Y. En cambio, la matriz, las filas son las X (vertical) y las columnas son las Y (horizontal).

Aquí, cuando separé todo en clases, observé y miré como se hacía para escribir en el canvas para reflejar los textos del canvas 2 o el texto que se muestra en pantIni. Esto se realiza de la siguiente forma:

```
this.cont.fillStyle = "#000000";      //definimos el color que
tendrá el string que pintemos
this.cont.font = "Bold 12px Verdana";    //definimos el tamaño
y el tipo de letra que escribiremos (está en negrita, por eso BOLD)
```

Acercándose ya a la versión final, iba viendo algunos fallos que se fueron solucionando para comprobar que todo fuera correctamente.

Uno de ellos era que cuando el personaje era asesinado por otro jugador, cuando se revivía, no se cambiaba la posición de la matriz y del vector, cosa que daba pie a errores que parecía que hubiese una batalla cuando no debía haberla.

Una de las últimas cosas que se fueron probando y cambiando para que funcionasen correctamente es que se pusieron las funciones draw, leerAccion,... de cada clase estado de juego, como funciones internas de la clase, es decir, “this.draw = function(){.....}”.

Para ir comprobando que las clases estado funcionasen, fui paulatinamente, clase por clase.

- Primero se comprobó la clase pantIni, que funcionase bien las teclas ahí en las clases, donde vi que había cometido el fallo de que no había vuelto a poner la tecla a false, con lo que se movía de forma errónea.
- Segundo, comprobé la clase pantHelp, que entrase y saliese de ella correctamente.
- Tercero, la clase pantGame, donde tube un pequeño problema, el cual venía ligado con la clase pantOut, ya que si salía del juego y posteriormente volvía a entrar en él, todo

estaba como antes de salir del juego, cosa que se arregló creando una instancia a la clase pantGame de nuevo.

- Cuarto, que el juego se pausase correctamente, es decir, que fuese de la clase pantGame a la pantPause y viceversa y que funcionase de manera correcta.

- Quinto, que cuando se muriese el jugador, ya fuera por muerte al pelear o por chocar contra un obstáculo, volviese a tener los parámetros iniciales del personaje.

Las últimas comprobaciones fueron:

- Pintar la cuadrícula del canvas para que se viera en todo momento en la posición en la que se encontraba.

- Aunque en el canvas 2 ya se mostraba la información del personaje activo, se modificó para que también se mostrase la imagen del personaje.

- En este punto, el final, es cuando decidí ponerle el sonido al juego. Obtuve la librería SoundManager2 y el fichero “sonido.js”.

5- Conclusiones

A la hora de escoger este proyecto, sabía que no sería fácil, puesto que no había hecho HTML y el javascript, aunque se parece en parte al Java, no es el mismo, es decir, que se podía decir de alguna manera como que empezaba de cero.

El resultado ha sido positivo, ya que he aprendido nuevas formas de programar y he aprendido una de las cosas que siempre me hacía pensar, el ¿Cómo funcionaban los videojuegos? ¿Cómo eran “por dentro”?

El proyecto ha servido también para conocer las novedades que el estándar HTML5 trae consigo y las mejoras que trae respecto a las anteriores versiones.

6- Recursos utilizados

Para realizar el proyecto, he utilizado diferentes recursos.

6.1- BIBLIOGRAFÍA:

Libros sobre HTML5 y Javascript que había en la biblioteca de la Universidad.

6.2- PAGINAS WEB:

<http://webea.net/99/guia-completa-de-html-5.html>
<http://www.moovas.net/posts/programacion/gu-a-b-sica-de-html5-1112.html>
<http://www.desarrolloweb.com/manuales/manual-canvas-html5.html>
<http://www.w3schools.com/js/default.asp>
http://www.w3schools.com/jsref/event_onkeydown.asp
<http://krook.org/jsdom/>
<http://www.desarrolloweb.com/articulos/439.php>

Y más paginas de las que no tengo guardadas las pestañas.

6.3- PROGRAMAS UTILIZADOS:

Para la realización del proyecto, se han utilizado diversos programas, algunos para Windows y otros para Mac OS X.

De los programas utilizados en Windows, en el que se distribuían las clases y se veían todos los ficheros era un programa llamado “Aptana 3”, y para comprobar el juego se utilizaba el navegador Google Chrome.

De los programas utilizados en Mac OS X está el Textmate, en el cual se ha escrito el código, aunque muy puntualmente se ha utilizado el Fraise. Para comprobar el juego que funcionase, se utilizaba también el navegador Google Chrome.

Para realizar el diagrama de clases se ha utilizado el programa “Dia” de Windows.

7- Anexos

7.1- Anexo 1: Navegadores

Debido a que se quería que el juego fuese multiplataforma, se ha comprobado que funcione en los diversos tipos de navegadores. Así, se ha probado con las siguientes versiones, con lo que de aquí en adelante, seguirá funcionando:

- Google Chrome: Versión 13.0.782.218
- Safari: Versión 5.1
- Mozilla Firefox: Versión 6.0.1
- Opera: Versión 11.51

Aunque parezca que debería ir en el resto de navegadores, hay un dato a tener en cuenta, y es el tema del sonido, puesto que no todos los navegadores soportan el mismo tipo de archivo de sonido. Aquí se ve un cuadro de los tipos de audio que soporta cada navegador.

Para saber que nombre corresponde a cada navegador, se proporciona también una pequeña lista, ya que son los motores de renderizado, que se encargan de renderizar el código e implementar el código multimedia.

- Google Chrome: WebKit
- Safari: WebKit
- Mozilla FireFox: Gecko
- Internet Explorer: Trident
- Opera: Presto

	WebKit	Gecko	Trident (IE9)	Presto
Mp3	*		*	
Ogg	* (Safari no)	*		*
Wav	* (Safari si)	*	*	*

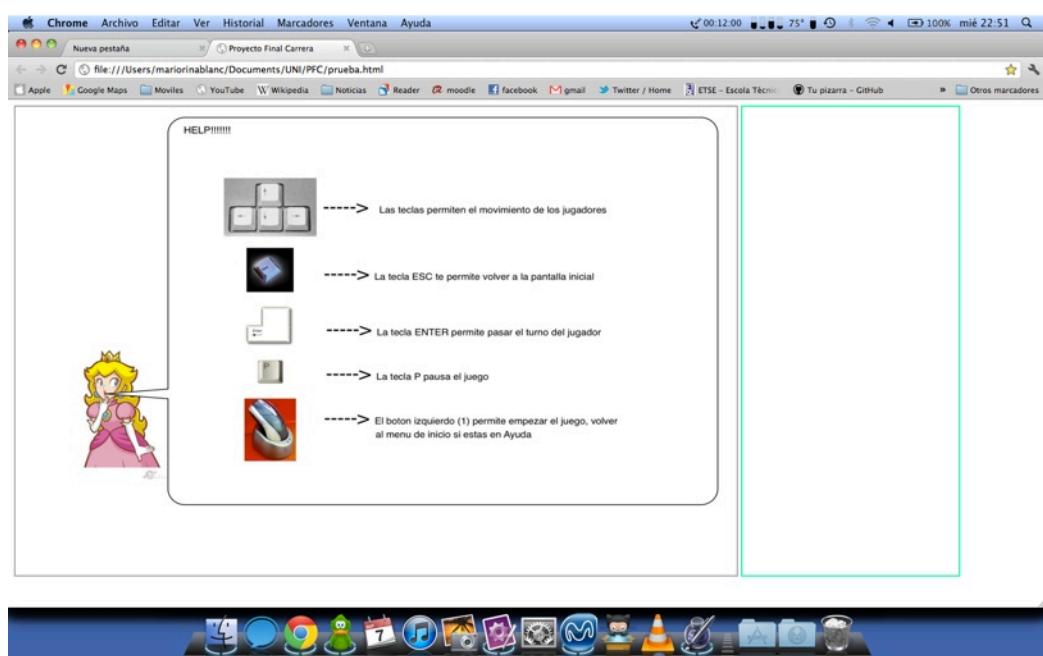
7.2- Anexo 2: Instrucciones

En este anexo se va a explicar un poco como se juega al videojuego que se ha creado. Para que quede mejor explicado, también se harán servir diversas capturas de pantalla del juego. Comencemos:



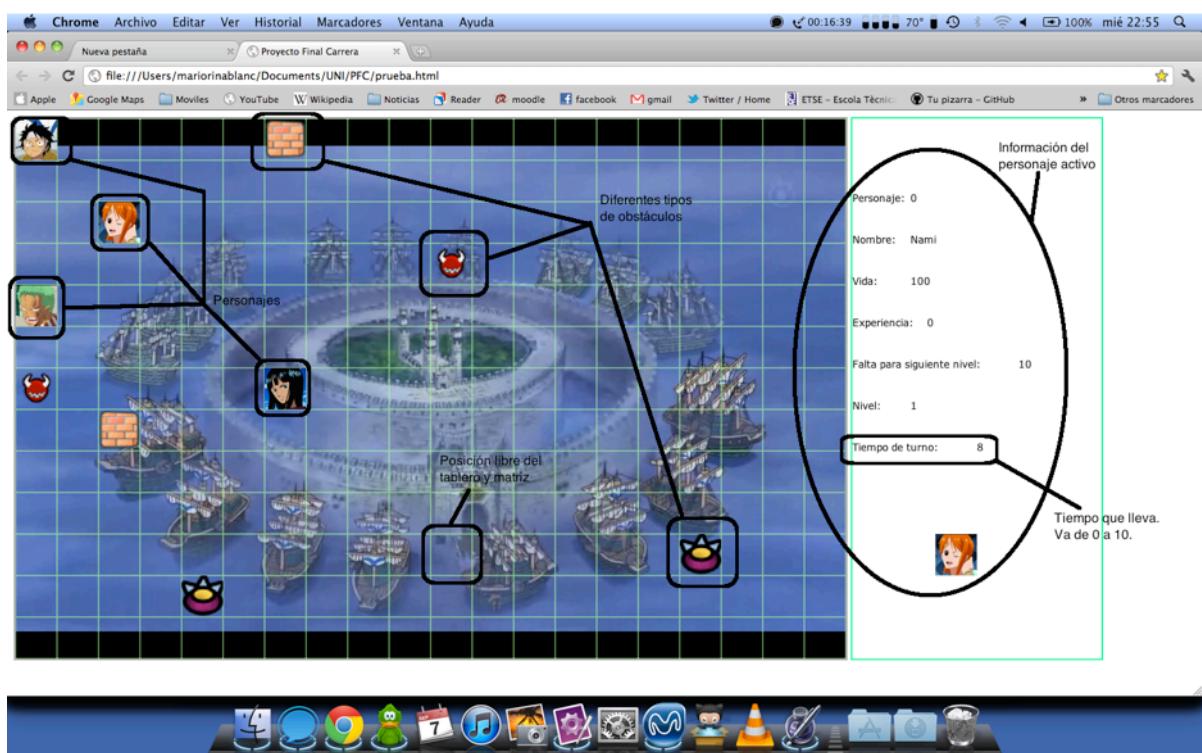
Aquí se ve la pantalla de inicio del juego, donde se ve que se pueden seleccionar 3 opciones: Jugar, Ayuda o Salir. Según se vaya pulsando las teclas de dirección “arriba” y “abajo”, la flecha que en la imagen apunta a Jugar, se movería a Ayuda o a Salir, dependiendo del número de veces que se pulsen las teclas y para acceder a una de las opciones, basta con pulsar el botón izquierdo del ratón.

Por ejemplo, vayamos a pulsar la opción de Ayuda:

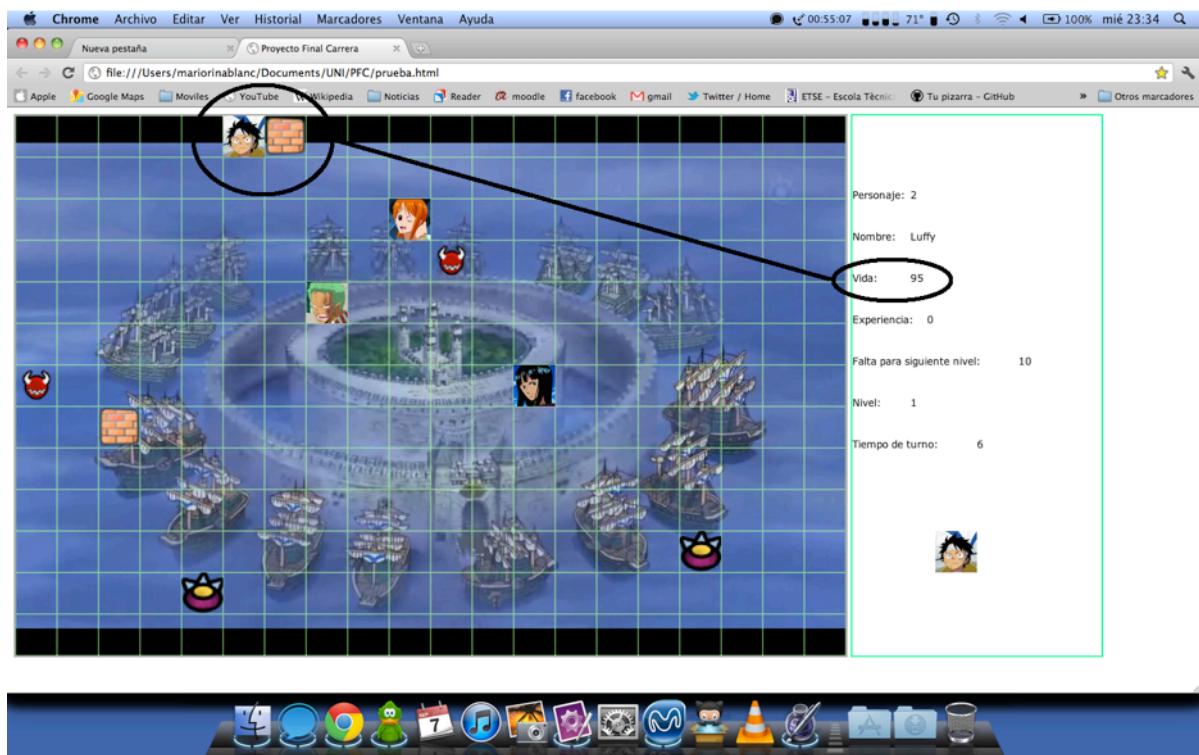


Aquí se pueden apreciar las diferentes teclas y botones que sirven durante el juego para realizar acciones, ya sea moverse, pausar el juego, saltar el turno, salir del juego,... Para volver al menú principal, el de la primera imagen, basta con pulsar el botón izquierdo del ratón otra vez.

Ahora vamos a fijarnos en el estado inicial del juego, es decir, cuando pulsas la opción de jugar. La siguiente imagen muestra el tablero inicial del juego.

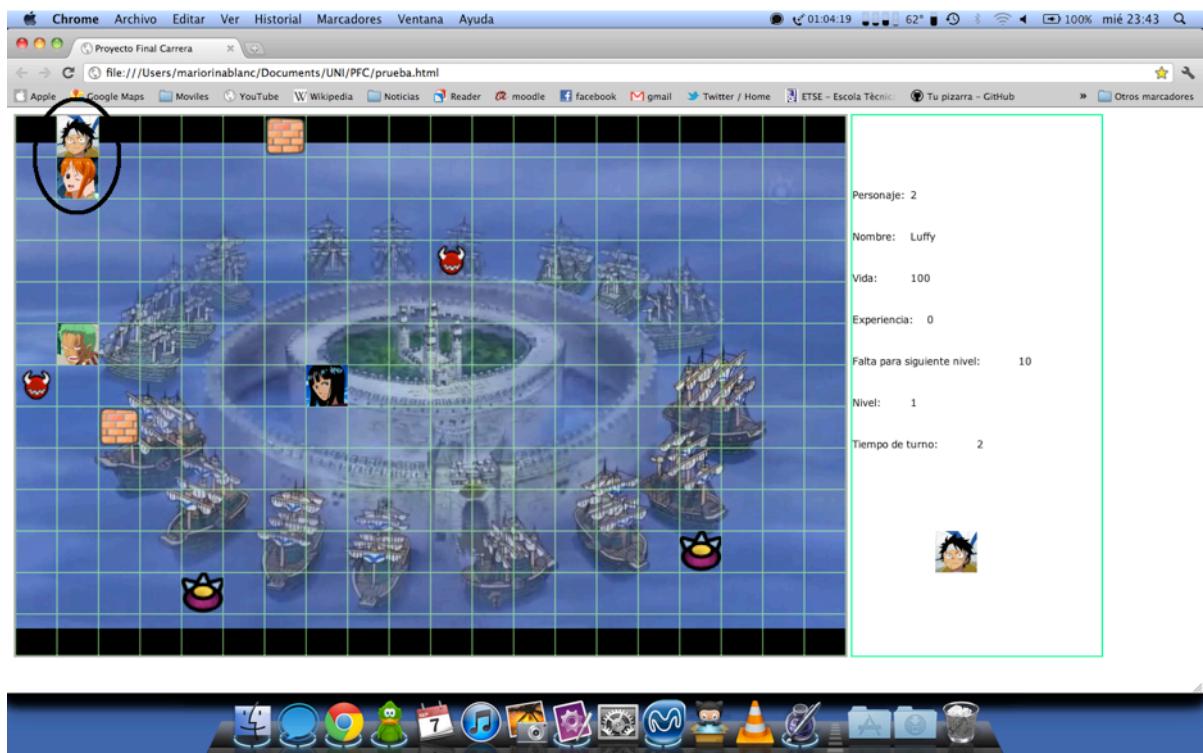


Como se puede observar, hay determinada información que se debe saber. Por ejemplo, las 4 imágenes que están unidas, son los personajes que juegan la partida. Y se mueven por las diferentes casillas del tablero. La casilla marcada solitaria, es una casilla vacía, donde los personajes pueden colocarse sin ningún tipo de peligro. Las casillas con las que hay que tener cuidado son las 3 casillas que están unidas, y sus respectivas iguales que no están marcadas, puesto que eso son los diferentes obstáculos del juego, que cada uno le quitará al personaje activo (cuya información se ve reflejada en el recuadro verde, que está marcado con el círculo) una determinada cantidad de vida.

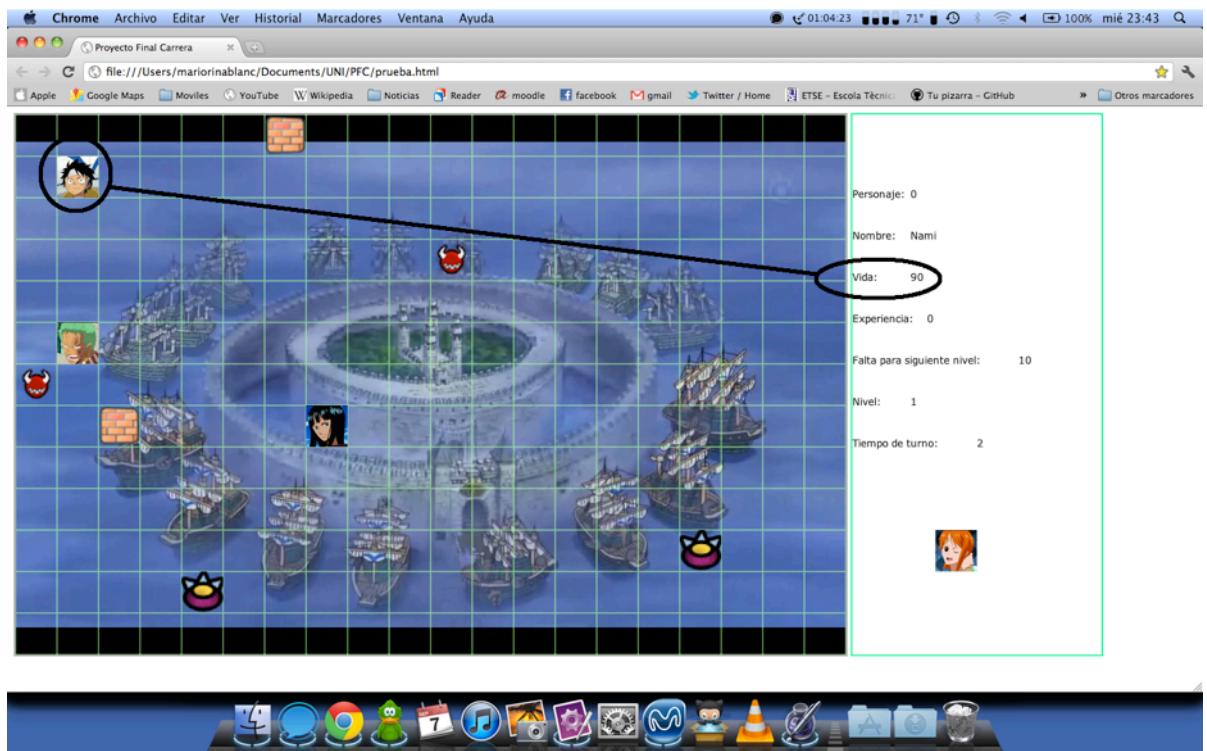


En esta imagen, se ve un ejemplo de que un personaje ha colisionado contra un obstáculo y, como se aprecia también (no se ve demasiado debido a que la imagen es algo pequeña) la vida inicial que tenía de 100 puntos, se ha visto reducida debido al impacto.

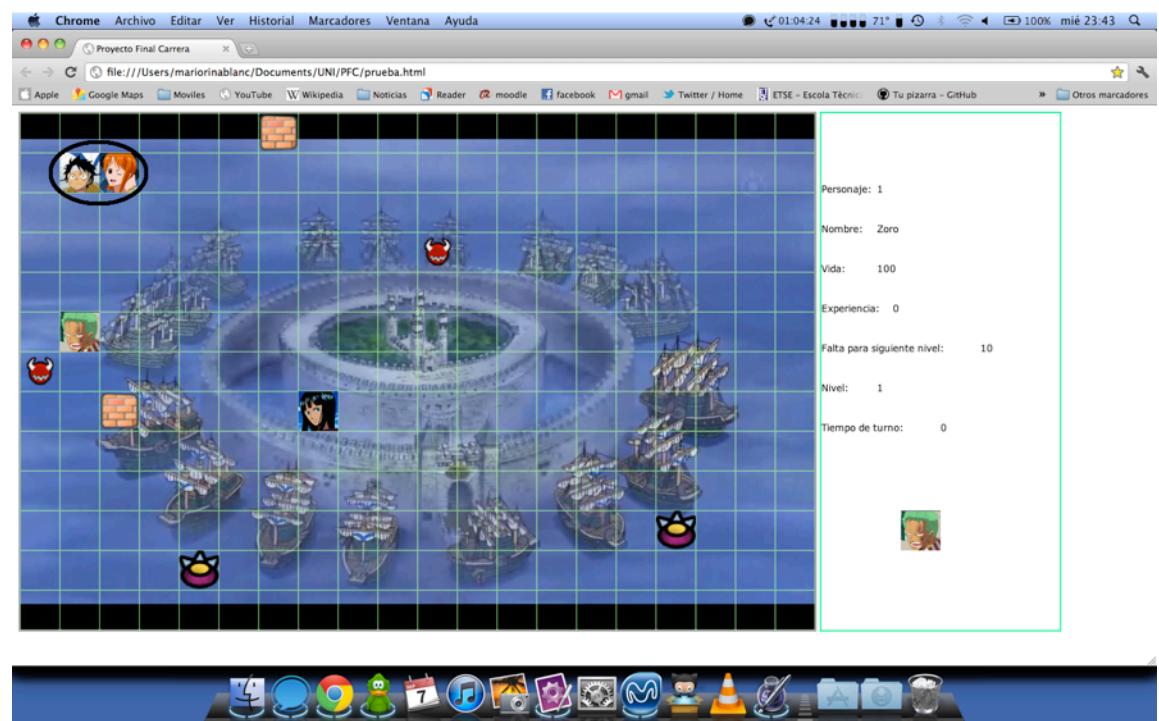
A continuación se verá un coche entre personajes. Lo que se va a poder observar es el instante previo a la batalla, cuando el personaje ha de hacer el movimiento y atacar (primera imagen):



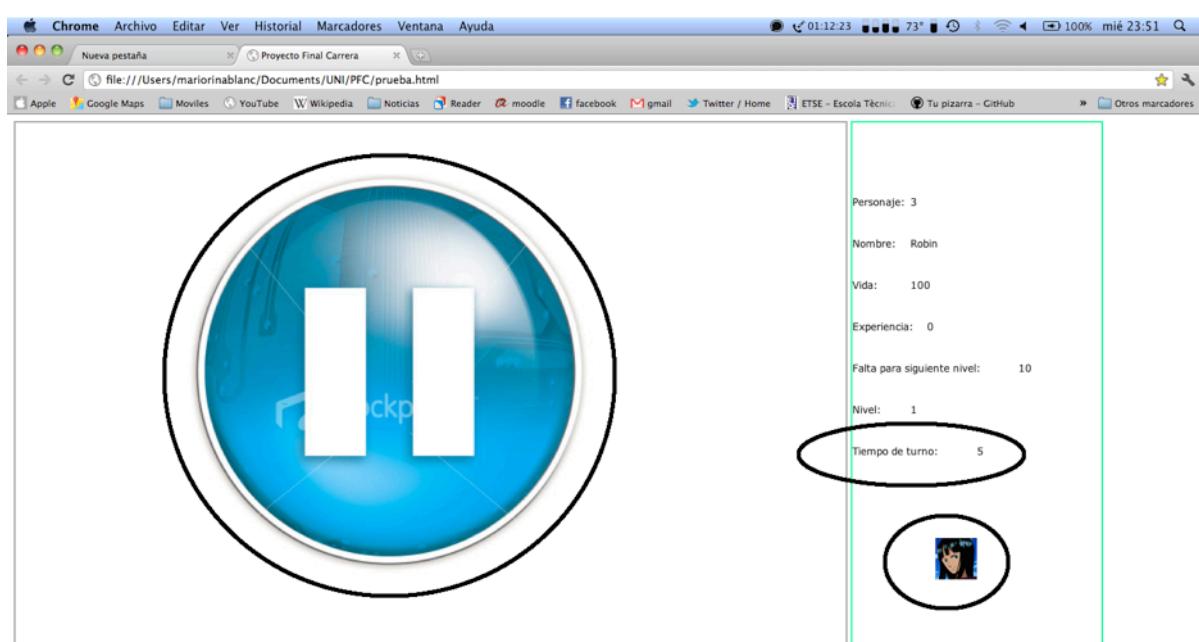
el instante en el que se ha hecho el movimiento, con lo que un personaje queda superpuesto al otro (segunda imagen), donde, aunque se ve demasiado pequeño y no se aprecia, es cuando el personaje atacado ha de realizar su movimiento (todavía está oculto) y se observa que la vida del personaje se ha visto disminuida por el ataque sufrido,

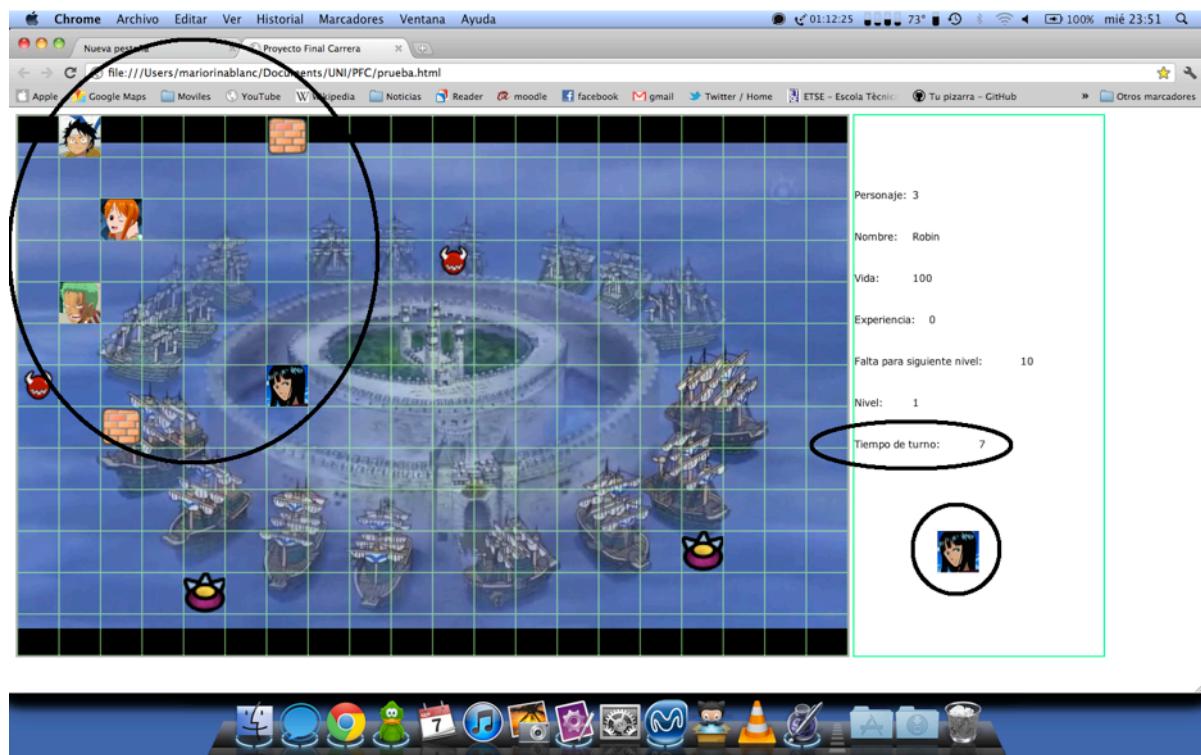


y por último, la tercera imagen, es justo después de que el personaje atacado haya hecho su movimiento.



Ahora se va a mostrar lo que pasa cuando el juego se pausa y se reanuda de nuevo. El ejemplo se verá cuando un personaje está realizando el movimiento (el tiempo pasa), cuando el juego es pausado, y cuando el juego es reanudado.





Como se aprecia, la primera imagen es el instante en el cual el jugador está activo y decide pausar el juego por alguna razón (imagen 2), donde se puede apreciar también que el personaje sigue ahí mostrándose con el tiempo, para indicar que el juego está pausado con ese personaje como personaje activo, y que lleva X segundos de turno. Por último, la tercera imagen muestra el momento en el que el juego ha sido reanudado. Como se puede apreciar, el jugador activo sigue siendo el mismo, el resto de los jugadores están en las mismas posiciones que antes y el tiempo no ha pasado (en la imagen se aprecia que ha pasado uno o dos segundos, pero mientras se hace la captura de pantalla no es exacto).

Para terminar la información sobre como se juega, se muestra la siguiente imagen, que es el resultado de pulsar la tecla ESC (como se indicaba en la pantalla de ayuda), que corresponde a la tecla que indica que el juego se ha terminado.



La propia imagen ya indica que es lo que ha ocurrido y lo que se ha de hacer a continuación.

Como se ha ido indicando en la pequeña introducción e instrucciones de como funciona el juego, se puede hacer un resumen:

- Para entrar en el juego, ayuda o salir desde el menú principal, hay que posicionarse en la opción mediante las teclas de dirección y pulsar el botón izquierdo del ratón.
- Para volver al menú principal ya sea desde el menú de ayuda o si se ha pulsado la opción de salir, se ha de pulsar también el botón izquierdo del ratón.
- Para Pausar el juego desde el propio juego una vez iniciado, hay que pulsar la tecla de la letra P. Para salir del modo de pausa y volver al juego, basta con volver a pulsar la misma tecla.
- Para salir del juego hay que tocar la tecla ESC.

Algún dato de interés sobre el juego es que los personajes se moverán y realizarán daño conforme al nivel que tengan. Así si un personaje está a nivel 1, se moverá una casilla, y realizará un ataque de 10 unidades. Sin embargo, un personaje de nivel 3, se moverá tres casillas y hará un ataque de 30 unidades.

Como también se ha dicho, para mover el personaje, se pulsan las diferentes teclas de dirección, teniendo en cuenta que no se puede salir de los límites marcados por el tablero.

Otro dato relevante es que si un personaje no sabe qué acción realizar y desea pasar el turno, deberá pulsar, como también se indica en el menú de ayuda, la tecla “enter”, que es la escogida para esta tarea.

8- Resúmenes

8.1- Resumen en ESPAÑOL

El proyecto que se ha realizado trata de crear un videojuego mediante el lenguaje de programación javascript valiéndose de las novedades que el estándar HTML5 trae consigo.

Como se puede observar, el proyecto funciona en varias plataformas web, como Google Chrome o Mozilla Firefox.

Como se puede apreciar, al inicio hay las 3 opciones típicas, jugar, ayuda (instrucciones) y salir del juego.

Una vez se comienza a jugar, se aprecia que el juego es en cuadrícula, con los jugadores pudiéndose mover en las cuatro posiciones, arriba, abajo, derecha e izquierda.

El juego consiste en que cada jugador debe ir avanzando por el tablero sin chocarse con los obstáculos que se encuentra por el camino y alcanzar al resto de los jugadores, así, al atacarlos, se ganará experiencia y se irá subiendo de nivel, con lo que el personaje hará más daño en sus ataques y se desplazará a mayor velocidad.

El juego se podrá pausar en cualquier momento del juego, quedando los jugadores y el tiempo “congelados” hasta que se vuelva a reanudar el juego.

Así mismo, si se decide terminar la partida y salir del juego para más tarde querer volver a empezar una nueva partida, bastaría con pulsar la tecla ESC, la cual te permite volver al menú principal.

El videojuego también cuenta con sonidos, lo cual hace que sea más entretenido de jugar y en el momento en el que un personaje sufra algún daño, otro sonido se lo hará saber.

En definitiva, este es el Proyecto Final de Carrera que se ha realizado.

8.2- Resumen en CATALÁN

El projecte que s'ha realitzat tracta de crear un videojoc mitjançant el llenguatge de programació javascript valent-se de les novetats que l'estàndard HTML5 comporta.

Com es pot observar, el projecte funciona en diverses plataformes web, com Google Chrome o Mozilla Firefox.

Tal com es pot apreciar, a l'inici hi ha les tres opcions típiques, jugar, ajuda (instruccions) i sortir del joc.

Un cop es comença a jugar, s'aprecia que el joc és en quadrícula, on els jugadors es poden moure en les quatre posicions, amunt, avall, dreta i esquerra.

El joc consisteix en que cada jugador ha d'anar avançant pel tauler sense xocar amb els obstacles que es troba pel camí i arribar a la resta dels jugadors, així, al atacar-los, es guanyarà experiència i s'anirà pujant de nivell, de manera que el personatge tindrà més poder en els seus atacs i es desplaçarà a major velocitat.

El joc es podrà pausar en qualsevol moment del joc, quedant els jugadors i el temps "congelats" fins que es torni a reanudar el joc.

També, si es decideix acabar la partida i sortir del joc per a més tard voler tornar a començar una nova partida, n'hi hauria prou amb prémer la tecla ESC, la qual et permet tornar al menú principal.

El videojoc també compta amb efectes sonors, la qual cosa fa que sigui més entretingut de jugar i en el moment en què un personatge xoqui amb algún tipus d'obstacle, un altre so l'hi farà saber.

En definitiva, aquest és el Projecte Final de Carrera que s'ha realitzat.

8.3- Resumen en INGLÉS

The project I've been done is create a game using the JavaScript programming language using the new features that brings the HTML5 standard.

As you can see, the project works in several web platforms such as Google Chrome or Mozilla Firefox.

As can be seen at the beginning there are three typical options, play, help (instructions) and exit the game.

When you start playing, we see that the game is on a grid, with players being able to move in all four positions, up, down, left and right.

The game is for each player to advance through the board without colliding with the obstacle at time you pursue the other players, because if you attack them, you will gain experience and level up, and the character will do more damage in their attack and move faster.

The game can be paused at any point in the game, leaving players and time "frozen" until you return to resume the game.

Also, if you decide to end the game and quit for later starting a new game, just press the ESC key, which takes you back to the main menu.

The game also has sounds, which makes it more fun to play and when a character is damaged, another sound will let you know.

In short, this is the Final Project that has been done.