Desenvolupament de casos pràctics

Xavier Garcia Rodríguez

Desenvolupament web en l'entorn client

Índex

Introducció

Un desenvolupador d'aplicacions no ha de conèixer, només, les paraules clau d'un llenguatge i les estructures bàsiques, sinó que ha de ser capaç d'utilitzar aquests coneixements per implementar aplicacions complexes i saber com integrar-les en serveis de tercers, tant per fer consultes a través de l'API d'un tercer com per integrar biblioteques externes en una aplicació o desenvolupar els propis servidors de proves.

En aquesta unitat, **"Desenvolupament de casos pràctics"**, aprendreu com utilitzar Node.js per crear dos servidors senzills i poder comprovar el correcte funcionament de les vostres aplicacions que requereixin interactuar amb serveis web. Per una banda s'implementarà un servidor RESTful per comprovar les crides AJAX, i per l'altra, s'implementarà un servidor de xat que permeti connectar múltiples usuaris simultàniament.

Seguidament, veureu com utilitzar la biblioteca de Google Maps per crear un mapa simple amb marcadors i finestres d'informació, que es combinarà –amb la connexió mitjanant AJAX– amb un servei web que ofereix dades obertes per mostrar la localització de tots els cinemes de la ciutat de Barcelona.

Per acabar, es farà un repàs pas a pas dels components necessaris per crear un joc en HTML5, incloent-hi com a exemple el codi complet del joc *IOC Invaders* que podeu descarregar, modificar i utilitzar com a base per als vostres propis jocs.

Com que aquesta unitat és completament pràctica, és imprescindible seguir tots els exemples pas a pas, amb excepció del desenvolupament del joc *IOC Invaders*, que és força complicat i només s'ha d'entendre com funciona.

Resultats d'aprenentatge

En finalitzar aquesta unitat, l'alumne/a:

1. Desenvolupa aplicacions web dinàmiques, reconeixent i aplicant mecanismes de comunicació asíncrona entre client i servidor.

- Avalua els avantatges i els inconvenients d'utilitzar mecanismes de comunicació asíncrona entre client i servidor web.
- Analitza els mecanismes disponibles per a l'establiment de la comunicació asíncrona.
- Utilitza els objectes relacionats.
- Identifica les seves propietats i els seus mètodes.
- Utilitza comunicació asíncrona en l'actualització dinàmica del document web.
- Utilitza diferents formats en l'enviament i en la recepció d'informació.
- Programa aplicacions web asíncrones de manera que funcionin en diferents navegadors.
- Classifica i analitza llibreries que facilitin la incorporació de les tecnologies d'actualització dinàmica a la programació de pàgines web.
- Crea i depura programes que utilitzin aquestes llibreries.

1. Desenvolupament de casos pràctics

Tenir una bona base dels coneixements teòrics és fonamental, però practicar aplicant aquests coneixements és el que us permetrà desenvolupar les vostres pròpies aplicacions per molt complicades que arribin a ser.

S'ha de tenir en compte també que, en programació, és indispensable saber cercar informació a internet, tant documentació sobre el llenguatge com sobre una biblioteca o API concreta o alguna tecnologia amb la qual hàgiu de treballar (com sensors o realitat virtual, per exemple).

Tenir uns coneixements amplis en programació us permetrà entendre com es poden adaptar fàcilment les biblioteques i les noves tecnologies a les vostres aplicacions, per crear un programa de xat, una guia turística amb mapes o un joc en HTML5.

Convé destacar que en el dia a dia dels desenvolupadors d'aplicacions és habitual haver d'utilitzar eines mitjançant la línia d'ordres (compiladors o preprocessadors de CSS) i que moltes d'aquestes eines estan desenvolupades en Node.js. És a dir, estan desenvolupades en JavaScript, però a la banda del servidor.

Conèixer el funcionament de Node.js permet als desenvolupadors crear les seves pròpies eines per a la línia d'ordres, així com desenvolupar servidors tan simples o complexos com sigui necessari per comprovar el funcionament tant d'aplicacions web com dels programes de xat o dels jocs multijugador.

També és molt habitual haver d'utilitzar biblioteques i API de tercers, com per exemple Google Maps o fonts de dades externes. Sovint, per accedir a aquestes API s'ha de crear un compte al lloc web dels proveïdors i per fer-lo servir poden requerir algun tipus de pagament. En el cas de Google molts dels seus serveis són gratuïts fins a un cert punt, però per superar la quota cal activar els pagaments.

Les dades obertes són accessibles per a tothom i l'únic inconvenient per implementar una aplicació que les utilitzi és que la documentació sigui disponible i que admetin CORS o JSONP.

Cal no oblidar el desenvolupament de jocs en HTML5, ja que amb la desaparició de Flash i Java dels navegadors tots els jocs del web han passat a estar desenvolupats en JavaScript. A més a més, amb les millores en la potència dels equips és possible desenvolupar jocs força complexos que utilitzen gràfics en 3D i, fins i tot, preparats per a realitat virtual.

Realitat virtual al navegador

WebVR (webvr.info) és una API de JavaScript que permet utilitzar dispositius de realitat virtual al navegador.

1.1 Desenvolupament amb Node.js

Node.js (nodejs.org) està basat en el motor V8 de JavaScript de Chrome i permet desenvolupar aplicacions en JavaScript que són executades al servidor. Gràcies a això es poden reaprofitar molts dels coneixements adquirits com a desenvolupadors d'aplicacions a la banda del client.

Podeu trobar la documentació de Node.js a l'enllaç següent: goo.gl/40ZGhQ.

Cal destacar que Node.js és una tecnologia relativament recent, però amb molta demanda. Entre les aplicacions més habituals desenvolupades amb Node.js es troben els servidors webs, servidors de xat, jocs multijugador i eines per a la línia d'ordres (com per exemple els preprocessadors de CSS).

1.1.1 Introducció a Node.js

Per desenvolupar una aplicació amb Node.js només cal tenir-lo instal·lat i un editor de text pla. Cal tenir en compte que les aplicacions programades són executades a servidors remots, per posar en marxa un servei o com a línia d'ordres; per consegüent, els missatges que a JavaScript es mostren a la consola de les eines de desenvolupador es mostraran per la terminal. És a dir, els missatges d'error o els missatges que utilitzeu per depurar amb el mètode `console.log` es mostraran a la vostra terminal.

S'ha de tenir en compte que a Node.js es programa amb JavaScript, però hi ha algunes diferències:

- No hi ha disponible cap de les funcions dels navegadors ni del DOM.
- El sistema de mòduls i requeriments és similar al d'ES6 (ES5 no té sistema de mòduls).
- Cada fitxer que es carrega amb `require` és un mòdul que encapsula totes les seves funcionalitats i només exposen alguns mètodes o propietats; no és el mateix que carregar múltiples fitxers JavaScript al navegador.

Hi ha prou similituds per desenvolupar algunes aplicacions senzilles, però es recomana consultar la documentació oficial de Node.js i els tutorials si voleu aprofundir en les seves possibilitats.

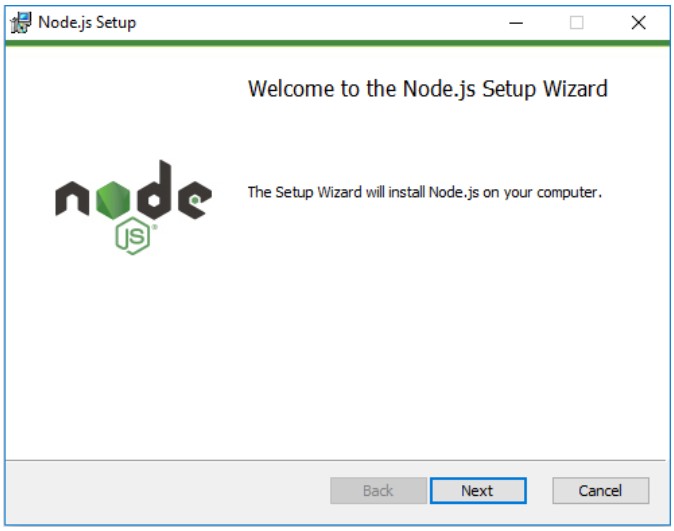
Instal·lació de Node.js

Abans de poder desenvolupar qualsevol aplicació amb Node.js cal instal·lar-lo. El procés és molt senzill en totes les plataformes, tot i que s'ha d'anar amb compte amb la versió que necessiteu: en alguns sistemes operatius, per defecte, s'instal·len versions molt desactualitzades.

Per instal·lar Node.js per a Windows s'han de seguir els passos següents:

- Entreu a la pàgina de descàrregues de Node.js: nodejs.org/en/download.
- Seleccioneu el vostre sistema operatiu: a la banda superior us apareixerà per defecte la darrera versió de l'instal·lable per a Windows 64 bits i Mac. En cas que vulgueu algun altre format, el podeu trobar a sota.
- Una vegada descarregat l'instal·lable per a Windows heu de fer doble clic sobre el fitxer i s'obrirà l'instal·lador, que es mostra a la figura 1.1.

FIGURA 1.1. Instal·lador de Node.js per a Windows

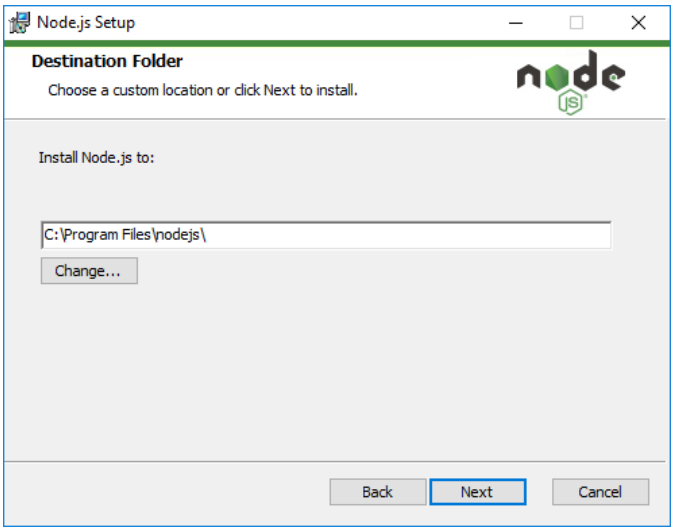


- S'ha d'acceptar la llicència d'ús, com es mostra a la figura 1.2.

FIGURA 1.2. Llicència de Node.js per a Windows

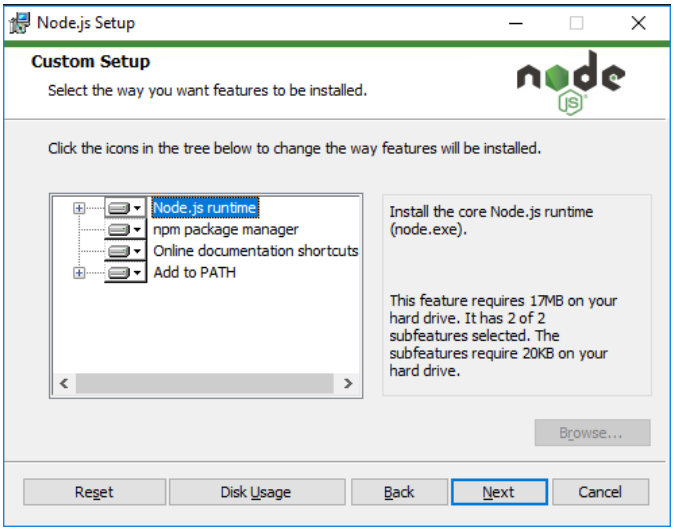
- Seguidament s'ha de seleccionar la carpeta de destí, com es mostra a la figura 1.3.

FIGURA 1.3. Selecció de la carpeta d'instal·lació de Node.js



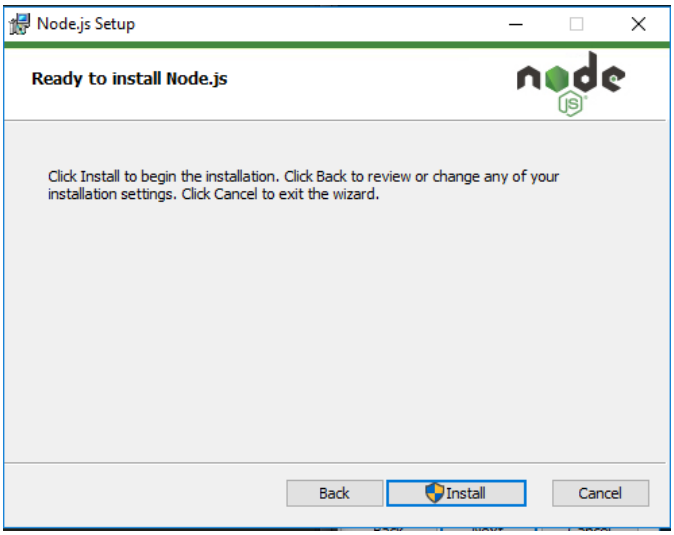
- El següent pas és seleccionar les característiques a instal·lar. L'instal·lador inclou el mateix Node.js, accessos directes a la documentació, l'instal·lador de paquets npm i la configuració automàtica de la ruta, tal com es mostra a la figura 1.4.

FIGURA 1.4. Selecció de característiques d'instal·lació de Node.js



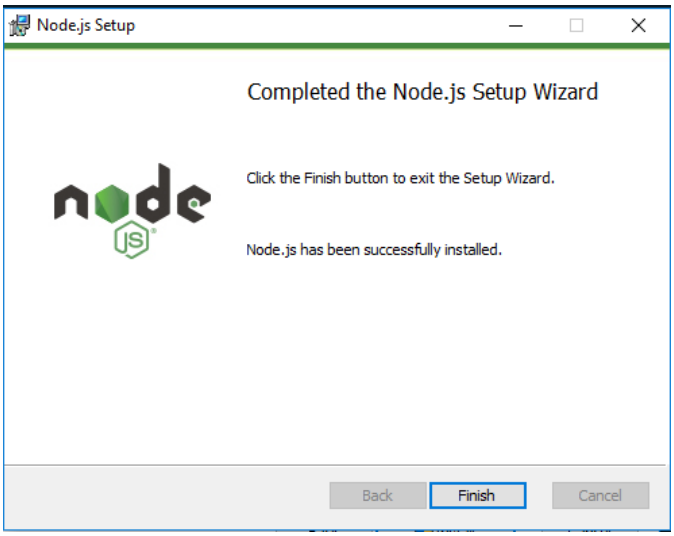
- L'últim pas és confirmar la instal·lació, com es mostra a la figura 1.5.

Figura 1.5. Confirmació d'instal·lació



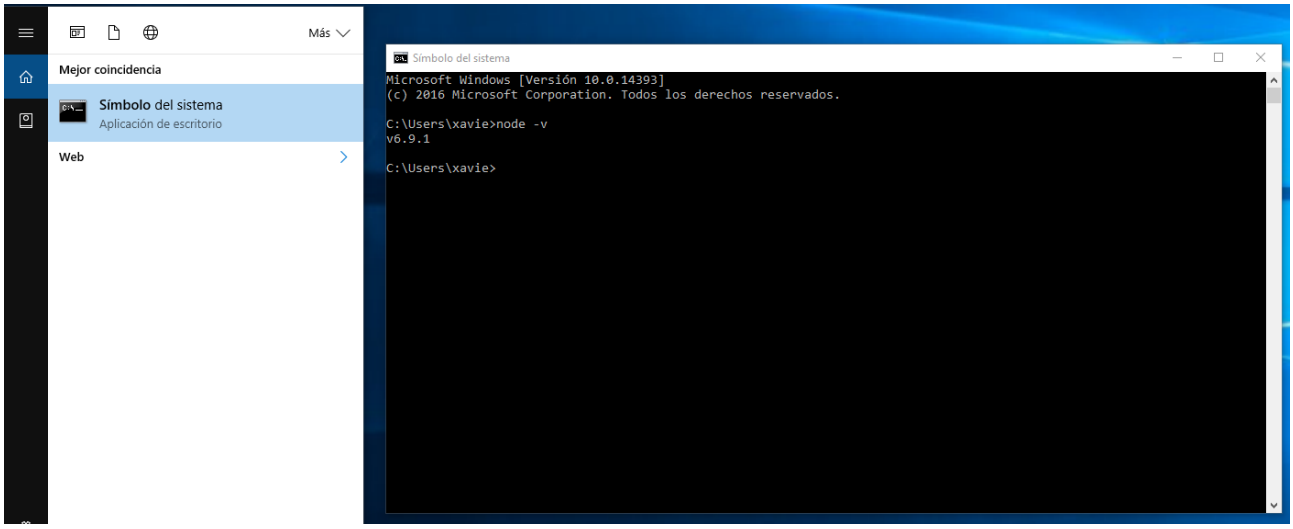
- Una vegada finalitzada la instal·lació rebreu la confirmació d'èxit, que es pot veure a la figura 1.6.

Figura 1.6. Finalització d'instal·lació



Per comprovar que s'ha instal·lat correctament heu d'accedir al símbol del sistema, com es pot apreciar a la figura 1.7, o a la terminal millorada de Windows (anomenada PowerShell).

FIGURA 1.7. Comprovació de la versió de Node.js al símbol del sistema



En tots dos casos amb l'ordre `node -v` se us ha de mostrar el número de la versió. En cas contrari s'haurà produït un error a la instal·lació o a la configuració; proveu de reinstal·lar-lo amb totes les opcions per defecte i pareu atenció a qualsevol possible missatge d'error.

Addicionalment s'haurà instal·lat el gestor de paquets npm. Per comprovar-ho podeu escriure a la terminal `npm -v` i us n'ha de mostrar la versió. Aquest gestor us permetrà instal·lar nous mòduls i actualitzar la versió de node i del mateix gestor de paquets.

npm

npm (www.npmjs.com) és, per defecte, el gestor de paquets per a Node.js. Alguns ho interpreten com a un acrònim de *Node Package Manager*.

Els usuaris de Mac, una vegada finalitzada la instal·lació, poden comprovar que s'ha instal·lat correctament escrivint a la terminal:

```
~ $ node –v
```

La instal·lació amb la distribució de Linux Ubuntu es pot fer utilitzant el gestor de paquets APT des de la terminal:

APT

APT, acrònim d'*Advanced Packaging Tool* (Eina Avançada d'Empaquetat), és un sistema de gestió de paquets creat pel projecte Debian (inclòs a Ubuntu) que permet instal·lar i eliminar programes mitjançant la línia d'ordres.

```
sudo apt–get update
sudo apt–get install nodejs
```

Cal destacar que a Ubuntu l'executable de node es diu `nodejs` per evitar conflictes amb altres paquets, en canvi, a Mac i Windows (o altres distribucions de Linux) el nom de l'executable és `node`.

Altres versions de Node.js a Ubuntu

La versió de node que s'instal·la utilitzant el gestor de paquets d'Ubuntu acostuma a estar molt desactualitzada. Si necessiteu fer servir una versió més recent, podeu seguir els passos descrits a l'enllaç següent: goo.gl/oXkiu9.

```
nodejs –v
```

Fixeu-vos que la instal·lació de Node.js mitjançant el gestor de paquets també inclou la instal·lació d'npm:

```
npm –v
```

1.1.2 Primer programa amb Node.js: Hola món

Els programes desenvolupats amb Node.js s'anomenen *mòduls* i en desenvoluparlos s'utilitzen altres mòduls que formen part de Node.js; com per exemple el mòdul `http`, per crear servidors que processin peticions HTTP.

Per altra banda, és habitual haver d'utilitzar codi de tercers, siguin mòduls simples, biblioteques senceres o *frameworks* (entorns de treball). En aquest cas, es parla de *dependències*, ja que el mòdul principal depèn d'aquests mòduls per funcionar.

'Framework' o entorn de treball

Un *framework*, en català 'entorn o marc de treball', és una infraestructura de programari que, en la programació orientada a objectes, facilita la concepció de les aplicacions mitjançant la utilització de biblioteques de classes o generadors de programes.

Creeu un fitxer anomenat `hola-mon.js`, dins d'un directori propi que servirà de contenidor per al vostre mòdul, amb el contingut següent:

```
1 var http = require('http');
2 http.createServer(function(peticio, resposta) {
3         resposta.writeHead(200, {'Content-Type':'text/plain;charset=utf-8'});
4         resposta.end('Hola món');
5 }).listen(8080, '127.0.0.1');
6 console.log('Servidor executant-se a http://127.0.0.1:8080/');
```

Tingueu en compte que, en cas que treballeu amb un servidor remot, en lloc de la IP 127.0.0.1 heu de posar la IP del vostre servidor.

Per posar-lo en marxa, haureu d'escriure a la terminal del sistema operatiu `node hola-mon.js` o, si treballeu amb Ubuntu, `nodejs hola-mon.js`.

Si entreu amb el navegador a l'adreça del servidor especificant el port 8080, veureu el missatge `Hola Món`.

Aplicacions web complexes

Per al desenvolupament d'aplicacions web complexes amb Node.js es recomana utilitzar el *framework* Express (expressjs.com).

Podeu trobar tota la documentació del mòdul `HTTP` a l'enllaç següent: goo.gl/C9GPNk.

A la primera línia es demana carregar el mòdul `http`. Aquest mòdul exposa el mètode `createServer`, que serveix per crear un nou servidor HTTP. Aquest servidor gestionarà les peticions rebudes.

Aquesta funció rebrà dos objectes com a paràmetres: la petició que s'ha enviat des del navegador (`http.ClientRequest`) i la resposta (`http.ServerResponse`), que serà retornada al client.

En aquest cas, s'utilitza el mètode `writeHead` de la resposta per escriure el codi de retorn a la capçalera i s'informa que s'ha processat correctament (200) i el tipus de contingut: un text pla codificat com a UTF-8. Això permet que els signes d'accentuació es mostrin correctament.

Seguidament, s'afegeix el cos de la resposta a enviar, les dades, que en aquest cas només inclouen la cadena `Hola món\n`. Si accediu a la pàgina des del navegador, probablement s'haurà afegit automàticament un codi HTML similar al següent:

```
1 <html>
2   <head>
3     <link rel="alternate stylesheet" type="text/css" href="resource://gre-
          resources/plaintext.css" title="Ajusta les línies llargues">
4   </head>
5   <body>
6     <pre>Hola Món</pre>
7   </body>
```

```
</html>
```

En canvi, si inspeccioneu la resposta, veureu que el seu contingut és `Hola Món`. Això es deu al fet que els navegadors, en rebre determinats tipus de continguts (en aquest cas text pla), el representen de la forma que consideren més adient.

Cal destacar que la funció `createServer` retorna un objecte de tipus `http.Server`, i és a partir d'aquest que s'invoca el mètode `listen`, que indica que s'han d'escoltar les peticions de l'adreça `127.0.0.1` pel port `8080`.

Un mètode alternatiu d'implementar el mateix exemple seria el següent:

```
var http = require('http');
var server = http.createServer();
server.on('request', function(peticio, resposta) {
        resposta.writeHead(200, {'Content-Type':'text/plain;charset=utf-8'});
        resposta.end('Adeu Món');
});
server.listen(8080,"127.0.0.1");
```

Fixeu-vos que en aquest cas primer s'ha creat el servidor sense passar cap paràmetre. Seguidament s'ha cridat el mètode `on` del servidor, que permet detectar diferents *events*; concretament es demana detectar l'*event* `request` (que és disparat en rebre una petició) i que es cridi la funció passada com a argument. Finalment s'indica que s'han d'escoltar les peticions pel port `8080` a l'adreça IP `127.0.0.1`.

1.1.3 Gestor de paquets npm

Node.js treballa amb un gestor de paquets que permet instal·lar nous mòduls molt fàcilment. A més a més, permet enregistrar els vostres mòduls al seu repositori públic, de manera que altres usuaris puguin trobar-los i utilitzar-los (si es vol restringir-los a determinats usuaris, cal un compte de pagament).

El gestor de paquets npm s'instal·la automàticament amb l'instal·lador de Node per a Windows i Mac; en canvi, a Linux cal instal·lar-lo individualment:

```
sudo apt-get install npm
```

Podeu trobar el cercador de mòduls d'npm a la seva pàgina principal: www.npmjs.com.

Una vegada instal·lat, es pot actualitzar així mateix, com si es tractés de qualsevol altre mòdul. Per exemple, a Windows i Mac es pot actualitzar així:

```
npm install npm -g
```

Mentre que a Linux s'han de demanar permisos d'administrador:

```
sudo npm install npm -g
```

En tots dos casos, l'opció `-g` indica que l'actualització s'ha d'aplicar globalment.

De la mateixa manera, per instal·lar qualsevol mòdul es fa servir la mateixa ordre. Per exemple, per instal·lar el mòdul `express` es faria de la manera següent:

```
1 npm install express
```

Si no s'especifica el nom del mòdul i es troba el fitxer `package.json`, s'instal·laran les dependències indicades en aquest fitxer. Gràcies a aquesta característica no heu d'incloure els mòduls i dependències externes (és a dir, la carpeta `node-modules`) al vostre sistema de control de versions.

Sistemes de control de versions

Els sistemes de control de versions, com GIT o Mercurial, permeten gestionar tots els canvis realitzats en un projecte. Podeu trobar més informació en l'enllaç següent: goo.gl/fo2T51.

De la mateixa manera, si descarregueu un programa de tercers, només haureu d'utilitzar l'ordre `npm install` perquè totes les dependències necessàries es descarreguin i s'instal·lin.

Per altra banda, quan es vol actualitzar un mòdul s'ha de fer servir l'ordre `update`. Per exemple, per actualitzar el mòdul `express` s'utilitzaria l'ordre següent:

```
1 npm update express
```

En cas de no especificar el nom del mòdul, s'actualitzaran tots els mòduls.

L'ordre que cal utilitzar per eliminar un mòdul és `remove`; aquesta ordre elimina tant el mòdul indicat com les seves dependències.

```
1 npm remove express
```

Cal destacar que, si s'utilitza sense indicar el mòdul que s'ha d'eliminar, es demanaran permisos d'administrador.

Per resumir, les ordres més importants del gestor de paquets npm són les següents:

- **install**: per instal·lar nous mòduls o actualitzar el mateix npm.
- **update**: per actualitzar mòduls.
- **remove**: per eliminar un mòdul instal·lat.
- **ls**: llista tots els mòduls.
- **-g**: per indicar que l'ordre s'aplica a l'entorn global.

Instal·lació global i local de mòduls

El gestor de paquets npm permet instal·lar els mòduls de forma global o local. La diferència és que si es fa localment, es copiarà dins del directori `node_modules` del mòdul i serà utilitzable només per aquest mòdul concret. En canvi, si s'utilitza l'opció `-g`, s'afegirà dins del directori `node_modules` del directori d'instal·lació de node i serà accessible globalment, és a dir, per a tots els projectes.

Com que el gestor de paquets npm s'ha d'utilitzar globalment, a l'hora d'actualitzar-lo sempre s'haurà de fer globalment. En canvi, quan requeriu un

mòdul per un projecte específic només cal afegir-lo a l'entorn local del projecte (per exemple, el mòdul `express`).

Altres mòduls que s'acostumen a instal·lar globalment són els preprocessadors de CSS, que converteixen el codi LESS o SASS en CSS, els automatitzadors com gulp i grunt o els compiladors d'ES6, que compilen el codi ES6 en ES5 (JavaScript 5), admès pràcticament per tots els navegadors.

gulp i grunt

gulp i grunt són dues eines que permeten automatitzar l'assemblatge d'una aplicació web: compilació d'ES6, precompilació de LESS o SAAS, optimització de fitxers CSS i JS.

Per llistar tots els mòduls instal·lats en un projecte es pot utilitzar l'ordre `node ls`, afegint la opció `-g` si es volen llistar els mòduls globals.

Descripció del mòdul: 'package.json'

Proveu d'afegir el mòdul `express` dins del directori on esteu treballant amb Node.js amb l'ordre següent:

```
1 npm install express
```

En acabar la instal·lació veureu els següents missatges d'advertència (lleugerament diferents segons el sistema operatiu utilitzat):

```
1 npm WARN enoent ENOENT: no such file or directory, open '/home/xavier/hola-mon/
      package.json'
2 npm WARN helloworld No description
3 npm WARN helloworld No repository field.
4 npm WARN helloworld No README data
5 npm WARN helloworld No license field.
```

La primera advertència es produeix perquè s'espera que el mòdul inclogui un fitxer anomenat `package.json` amb la informació del mòdul, i no l'hem afegit. La resta d'avisos es produeixen per la mateixa raó: com que no troba el fitxer, no pot llegir cap dels camps que es requereixen.

Aquest fitxer es pot generar manualment amb qualsevol editor de text pla o es pot inicialitzar amb l'ordre `npm init` (si premeu enter, agafarà el valor mostrat entre parèntesis):

```
1  xavier@Ubuntu-Node:~/hola-mon$ npm init
2  This utility will walk you through creating a package.json file.
3  It only covers the most common items, and tries to guess sensible defaults.
4
5  See 'npm help json' for definitive documentation on these fields
6  and exactly what they do.
7
8  Use 'npm install <pkg> --save' afterwards to install a package and
9  save it as a dependency in the package.json file.
10
11 Press ^C at any time to quit.
12 name: (hola-mon)
13 version: (1.0.0)
14 description: Primer programa amb Node.js
15 entry point: (hola-mon.js)
16 test command:
17 git repository:
18 keywords:
19 author: Xavier Garcia
20 license: (ISC)
21 About to write to /home/xavier/hola-mon/package.json:
```

```
{
  "name": "hola-mon",
  "version": "1.0.0",
  "description": "Primer programa amb Node.js",
  "main": "hola-mon.js",
  "dependencies": {
    "express": "^4.14.0"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "el vostre nom",
  "license": "ISC"
}


Is this ok? (yes)
```

El resultat és el mateix que si creeu un fitxer manualment amb el contingut en format JSON:

```
{
  "name": "hola-mon",
  "version": "1.0.0",
  "description": "Primer programa amb Node.js",
  "main": "hola-mon.js",
  "dependencies": {
    "express": "^4.14.0"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "el vostre nom",
  "license": "ISC"
}
```

Si proveu d'instal·lar el mòdul `express` (tot i que ja es troba instal·lat), veureu que han desaparegut tots els avisos, excepte un que indica que no s'ha especificat cap camp per al repositori. Aquest avís el podeu ignorar perquè només és rellevant si publiqueu el vostre codi i feu servir un sistema de control de versions.

El fitxer 'package.json'

Podeu trobar tota la documentació sobre el fitxer `package.json` a l'enllaç següent: goo.gl/Q0HlTj.

Com es pot veure, en el fitxer generat s'hi afegeix la següent informació del mòdul:

- **name**: nom del mòdul.
- **version**: número de la versió.
- **description**: descripció del mòdul.
- **main**: fitxer principal del mòdul.
- **dependencies**: llista de mòduls amb la corresponent versió que utilitza aquest mòdul. Aquesta informació permet instal·lar un mòdul mitjançant npm i, alhora, que es descarreguin les seves dependències en la versió apropiada.
- **scripts**: aquesta és una propietat especial que permet afegir scripts specials; en aquest cas l'escript s'executaria en cridar l'ordre `npm test`. Podeu trobar-ne més informació a l'enllaç següent: goo.gl/QmpQ0s.

- **license**: llicència que s'aplica al vostre mòdul.
- **author**: nom del desenvolupador o desenvolupadors. Aquest camp admet més informació: per exemple, es pot utilitzar el format següent per indicar el nom i el correu electrònic del desenvolupador:

```
{
  "name": "el vostre nom",
  "email": "el vostre correu electrònic"
}
```

Cal tenir en compte que en eliminar un mòdul amb `npm remove`, no s'elimina del llistat de dependències, sinó que s'ha d'editar el fitxer `package.json` i eliminar manualment la línia que correspongui.

1.1.4 Exemples avançats

Com us podeu imaginar, crear un servidor que mostri sempre el mateix missatge a l'usuari no és gaire útil, normalment serà necessari analitzar quin és l'URL demanat al servidor i el mètode emprat per fer la petició (`GET`, `POST`, `PUT`...).

Combinant l'URL, el mètode d'enviament i els paràmetres es pot implementar un encaminador (*router*, en anglès), un component que s'encarregui de gestionar quin component generarà la resposta de la petició.

Cal tenir en compte que Node.js, al contrari que el JavaScript executat al navegador, sí que pot llegir i escriure del disc del servidor. Així doncs, es poden realitzar operacions de lectura i escriptura, per exemple, per generar un registre d'errors o d'usuaris connectats.

Obtenció de l'URL i el mètode

Obtenir l'URL i el mètode de la petició és molt senzill. Quan es dispara l'*event* `request` al servidor, s'invoca la funció associada a la detecció de l'*event*, que rep com a primer paràmetre un objecte. Aquest conté la informació de la petició i, com a segon paràmetre, un altre objecte que permet gestionar la resposta. A partir de l'objecte que conté la informació de la petició es pot extreure tant el mètode utilitzat com l'URL de la petició. A continuació podeu veure un exemple:

```
var http = require('http');
var server = http.createServer();
server.on('request', function(peticio, resposta) {
  var metode = peticio.method;
  var url = peticio.url;
  resposta.writeHead(200, {'Content-Type':'text/plain;charset=utf-8'});
  resposta.end('Rebuda petició per l\'URL ' + url + ' i el mètode ' + metode);
});
server.listen(8080,"127.0.0.1");
```

Si proveu d'accedir des del vostre navegador a l'adreça `127.0.0.1:8080/hola` veureu per pantalla el missatge següent:

```
Rebuda petició per l'URL /hola i el mètode GET
```

Com es pot apreciar, l'objecte `peticio` (que és una instància d'`http.ClientRequest`) permet accedir tant al mètode d'enviament com a l'URL a través dels mètodes `method` i `url` respectivament.

A partir d'aquests dos paràmetres es podria implementar un servei web REST molt simple, ja que només cal discriminar l'URL i el mètode de la petició per portar a terme una acció o una altra.

Per comprovar el funcionament amb el mètode `POST`, podeu crear un fitxer nou anomenat `formulari-node.html` amb el codi següent:

```
<html>
<body>
  <select id="metode">
    <option value="GET" selected>GET</option>
    <option value="POST">POST</option>
    <option value="PATCH">PATCH</option>
    <option value="DELETE">DELETE</option>
  </select>
  <button id="peticio">Enviar Petició</button>
  <p>Resposta:</p>
  <div id="resposta"></div>

  <script>
    var boto = document.getElementById('peticio');
    var resposta = document.getElementById('resposta');
    var metode = document.getElementById('metode');

    boto.addEventListener("click", function() {
      var httpRequest = new XMLHttpRequest();

      httpRequest.open(metode.value, 'http://127.0.0.1:8080/prova', true);
      httpRequest.send(null);

      httpRequest.onload = function () {
        resposta.innerHTML = httpRequest.responseText;
      }
    });
  </script>
</body>
</html>
```

Com es pot apreciar, s'ha creat una llista desplegable amb HTML que permet seleccionar el mètode d'enviament i un botó per realitzar la petició. Una vegada es fa clic al botó s'envia una petició AJAX, utilitzant el mètode seleccionat a la llista, a l'adreça `127.0.0.1`, port `8080` i URL `prova`.

Una vegada arriba la resposta es dispara l'*event* `load` i es crida la funció assignada a `onload`. Aquesta funció estableix com a contingut HTML la resposta en mode text. Fixeu-vos que no cal sobreescriure el tipus de contingut perquè el servidor l'enviarà correctament.

Malauradament, en molts casos aquest exemple no funcionarà a causa de la 'política del mateix origen' (*same-origin policy*). Com que en aquest cas teniu accés directament a l'aplicació del servidor, només cal incloure els mecanismes

CORS adequats per permetre l'accés des de qualsevol adreça i utilitzant qualsevol mètode:

Política del mateix origen

Es tracta d'una normativa que implementen tots els navegadors; podeu trobar-ne més informació a l'enllaç següent: goo.gl/qMDYhF.

```
var http = require('http');
var server = http.createServer();
server.on('request', function(peticio, resposta) {
  var metode = peticio.method;
  var url = peticio.url;

  resposta.setHeader('Access-Control-Allow-Origin', '*');
  resposta.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT,
      PATCH, DELETE');
  resposta.writeHead(200, {'Content-Type':'text/plain;charset=utf-8'});

  resposta.end('Rebuda petició per l\'URL ' + url + ' i el mètode ' + metode);
});
server.listen(8080,"127.0.0.1");
```

En aquesta versió, el servidor afegeix a la capçalera de la resposta el paràmetre `Access-Control-Allow-Origin=*`, fet que permet realitzar peticions AJAX des de qualsevol servidor; i el paràmetre `Access-Control-Allow-Mehtods=GET, POST, OPTIONS, PUT, PATCH, DELETE`, de manera que permet fer servir directament tots els mètodes HTTP.

Fixeu-vos en una diferència important: s'ha fet servir el mètode `setHeader` en lloc de `writeHead`. Aquest últim el que fa és escriure la capçalera amb el codi de resposta i el contingut passat com a paràmetre (opcionalment), de manera que no es pot modificar; en canvi, el mètode `setHeader` permet afegir paràmetres a la capçalera sense escriure-la, de manera que es poden afegir tants paràmetres com sigui necessari (per exemple, pot ser que algunes capçaleres només calgui enviar-les en alguns casos concrets).

Obtenció de paràmetres

Per obtenir els paràmetres de la petició teniu dues opcions: realitzar una implementació pròpia per dividir la cadena formada pels paràmetres o utilitzar mòduls de Node.js `url` i `querystring`. El primer d'aquests mòduls permet descompondre un URL en fragments i el segon converteix una cadena de consulta en un objecte de JavaScript.

En l'exemple següent podeu veure com s'han afegit dos paràmetres a la petició i com es fa el tractament dels dos casos possibles: l'enviament utilitzant el mètode `GET` (paràmetres codificats en l'URL) o els altres mètodes (paràmetres integrats al cos de la petició).

Primerament s'afegeixen dos camps de text per enviar la informació al servidor. Fixeu-vos que s'ha de fer un tractament especial en el cas del mètode `GET`, ja que les dades s'envien afegint-les a l'URL i no pas com a dades:

```
<html>

<body>
  <h1>Petició</h1>
  <select id="metode">
    <option value="GET">GET</option>
```

```
    <option value="POST" selected>POST</option>
    <option value="PATCH">PATCH</option>
    <option value="DELETE">DELETE</option>
  </select>
  <label>Nom:<input type="text" id="nom" /></label>
  <label>Cognom:<input type="text" id="cognom" /></label>
  <button id="peticio">Enviar Petició</button>
  <h1>Resposta</h1>
  <div id="resposta"></div>

  <script>
    var boto = document.getElementById('peticio');
    var resposta = document.getElementById('resposta');
    var metode = document.getElementById('metode');
    var nom = document.getElementById('nom');
    var cognom = document.getElementById('cognom');

    boto.addEventListener("click", function(e) {
      var url = "http://127.0.0.1:8080/proves";
      var dades = "nom=" + encodeURI(nom.value);
      dades += "&cognom=" + encodeURI(cognom.value);

      if (metode.value == "GET") {
        url +="?" + dades;
        dades = null;
      }

      var httpRequest = new XMLHttpRequest();
      httpRequest.open(metode.value, url, true);
      httpRequest.send(dades);
      httpRequest.onload = function (e) {
        resposta.innerHTML = httpRequest.responseText;
      }
    });
  </script>
</body>

</html>
```

Pel que fa al codi del servidor cal afegir-hi més canvis. Per una banda, s'han carregat els mòduls `url` i `querystring` per analitzar els URL i les cadenes de consulta respectivament; i per altra, s'ha de processar el cos del missatge per obtenir els paràmetres en el cas dels mètodes diferents de `GET`.

```
var http = require('http');
var queryString = require('querystring');
var url = require('url');

var server = http.createServer();
server.on('request', function(peticio, resposta) {
  resposta.setHeader('Access-Control-Allow-Origin', '*');
  resposta.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT,
      PATCH, DELETE');
  resposta.writeHead(200, {'Content-Type':'text/html;charset=utf-8'});

  var cos = '';
  peticio.on('data', function(dades) {
    if (cos.length>1e6) { // Limita la mida a 1.000.000 de bytes = 1e6
      peticio.connection.destroy();
    } else {
      cos += dades;
    }

  }).on('end', function() {
    var params;
    var dades;

    if (peticio.method === 'GET') {
```

```
24        dades = url.parse(peticio.url).query;
25      } else {
26        dades = cos;
27      }
28
29      params = queryString.parse(dades);
30
31      resposta.write('Rebuda petició per l\'URL ' + peticio.url + ' i el mètode '
            + peticio.method +"<br>");
32      resposta.write("<b>Nom:</b> " + decodeURI(params.nom) + "<br>");
33      resposta.write("<b>Cognom:</b> " + decodeURI(params.cognom) + "<br>");
34      resposta.end();
35    });
36  });
37
38  server.listen(8080,"127.0.0.1");
```

S'ha de tenir en compte que el cos de la petició s'obté d'un flux de dades, ja que la petició pot incloure elements molt pesats (per exemple, la pujada de fitxers o imatges) i no es llegeix d'una tacada. Per aquesta raó cal detectar l'*event* `data` de la petició, que indica que han arribat noves dades, i concatenar-la per obtenir el cos complet i l'*event* `end`, que indica que s'ha llegit completament la petició.

Quan la resposta conté codi HTML, és més adient utilitzar com a `Content-Type` el tipus `text/html`.

En aquest exemple s'ha afegit una mesura de seguretat extra: es comprova la mida del cos actual i, si supera el milió de bytes, interromp la connexió. En cas contrari, concatena el nou fragment al cos fins que es detecta l'*event* `end`, que indica que ha finalitzat la recepció de la petició.

Una vegada s'han obtingut totes les dades cal processar-les perquè els paràmetres arriben com una cadena de consulta. En aquest cas també cal distingir entre el mètode `GET` i la resta, ja que el mètode GET inclou la cadena de consulta a l'URL. Per obtenir-la s'ha d'analitzar i accedir a la propietat `query`, com es pot veure en aquest exemple: `url.parse(peticio.url).query`.

Per altra banda, en el cas dels mètodes `POST`, `PATCH`, `DELETE`, etc., la cadena de consulta és el cos del missatge. Una vegada s'ha obtingut la cadena de consulta, d'una manera o altra, cal analitzar-la utilitzant el mòdul `querystring`, com es pot veure a l'exemple següent: `params = queryString.parse(dades)`.

Una vegada obtinguts els paràmetres com un objecte de JavaScript, ja s'hi pot treballar accedint als valors i utilitzant la notació de claudàtors o la notació de punts (per exemple: `params.nom`).

Cal destacar que, tant en el fitxer del client com del servidor, s'han codificat i descodificat els paràmetres utilitzant les funcions `encodeURI` i `decodeURI` respectivament; cal fer-ho així per evitar problemes en incloure caràcters que poden corrompre la informació.

Fixeu-vos que, en aquest exemple, en lloc d'enviar el contingut de la resposta utilitzant el mètode `end`, s'ha afegit per parts utilitzant el mètode `write`. La diferència principal entre l'un i l'altre (com passa amb `setHeader` i `writeHead`) és que una vegada s'ha invocat el mètode `end` ja no es pot afegir res més al cos de la resposta; en canvi, es pot invocar el mètode `write` tantes vegades com sigui necessari per construir la resposta.

Encaminament ('routing')

En aquest context, el terme *encaminament* es refereix a l'acció que s'ha de realitzar (o pàgina que s'ha de carregar) segons l'URL i el mètode d'enviament d'una petició. Per realitzar aquest encaminament, en els casos més simples es pot implementar una solució pròpia, tot i que és recomanable fer servir un *framework* com Express o el mòdul de tercers `router`.

A continuació podeu veure un exemple, amb una implementació pròpia, que retorna diferents respostes segons l'URL seleccionat del menú desplegable. Per una banda, s'ha de generar un fitxer HTML que s'executa al navegador del client, i per altra, el codi del servidor HTPP de Node.js.

```
<html>
<body>
  <h1>Petició</h1>
  <select id="metode">
    <option value="GET">GET</option>
    <option value="POST" selected>POST</option>
    <option value="PATCH">PATCH</option>
    <option value="DELETE">DELETE</option>
  </select>
<select id="provincia">
    <option value="Barcelona" selected>Barcelona</option>
    <option value="Girona" selected>Girona</option>
    <option value="Lleida">Lleida</option>
    <option value="Tarragona">Tarragona</option>
  </select>

  <button id="peticio">Enviar Petició</button>
  <h1>Resposta</h1>
  <div id="resposta"></div>
  <script>
    var boto = document.getElementById('peticio');
    var resposta = document.getElementById('resposta');
    var metode = document.getElementById('metode');
    var provincia = document.getElementById('provincia');

    boto.addEventListener("click", function(e) {
      var url = "http://127.0.0.1:8080/" + provincia.value;
      var httpRequest = new XMLHttpRequest();

      httpRequest.open(metode.value, url, true);
      httpRequest.send(null);

      httpRequest.onload = function (e) {
        resposta.innerHTML = httpRequest.responseText;
      }
      });
  </script>
</body>
</html>
```

En aquest cas, com que no s'utilitzaran paràmetres, només cal especificar el mètode i l'URL de destí. Per aquest mateix motiu no cal carregar el mòdul `querystring`, però sí que és necessari carregar el mòdul `url` per obtenir la ruta que s'ha de processar.

```
var http = require('http');
var url = require('url');

var server = http.createServer();
server.on('request', function(peticio, resposta) {
```

```
  resposta.setHeader('Access-Control-Allow-Origin', '*');
  resposta.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');
  resposta.writeHead(200, {'Content-Type':'text/html;charset=utf-8'});

  var urlEncaminament = url.parse(peticio.url).path;
  encaminar(urlEncaminament, peticio.method, resposta);
  resposta.end();
});

server.listen(8080,"127.0.0.1");

function encaminar(url, metode, resposta) {
  var vista;

  switch (url) {
    case '/Barcelona':
      vista = controladorBarcelona(metode);
      break;

    case '/Girona':
      vista = controladorGirona(metode);
      break;

    case '/Lleida':
      vista =  controladorLleida(metode);
      break;

    case '/Tarragona':
      vista = controladorTarragona(metode);
      break;
    }

    resposta.write(vista);
}


function controladorBarcelona(metode) {
  var vista = "<h2>Barcelona</h2>";

    switch (metode) {
      case 'GET':
        vista += "Simulació: dades de Barcelona obtingudes";
        break;

      case 'POST':
        vista += "Simulació: dades de Barcelona afegides";
        break;

      case 'DELETE':
        vista += "Simulació: dades de Barcelona eliminades";
        break;

      case 'PATCH':
        vista += "Simulació: dades de Barcelona actualitzades";
        break;
    }

    return vista;
}

function controladorGirona(metode) {
        return "<h2>Girona</h2>No hi ha cap acció especial per a <b>Girona</b>";
}

function controladorLleida(metode) {
        var vista = "<h2>Lleida</h2>";
        if (metode === 'GET') {
                vista += "A Lleida s'envia un missatge específic si s'utilitza
```

```
                    el mètode GET";
74          } else {
75                  vista += "A Lleida tots els mètodes excepte GET retornen un
                        missatge genèric";
76          }
77
78          return vista;
79  }
80
81  function controladorTarragona(metode)  {
82          return "<h2>Tarragona</h2>Tampoc s'ha implementat cap acció especial
                per a <b>Tarragona</b>";
83  }
```

Una vegada rebuda la petició, s'invoca la funció `encaminar`, passant com a paràmetres el mètode, la resposta i la ruta de l'URL. Dintre de la funció, segons el valor de la ruta, s'invocarà una funció o una altra, que farà el paper de controlador. Aquests controladors s'encarreguen de generar una vista (en aquest cas es tracta simplement d'una cadena de text formatat com a HTML) que s'escriu a la resposta.

S'ha fet servir `controlador` com a nom de les funcions, perquè és el funcionament habitual d'aquest tipus d'aplicacions. L'encaminador invoca els mètodes corresponents segons el mètode i ruta rebuts, i retorna o envia la vista generada. En aquest cas, com que no existeix cap model, la vista es genera directament fent servir textos estàtics.

Habitualment els sistemes d'encaminament accepten paràmetres com a cadena de consulta (tant al cos del missatge com incrustats a l'URL) i com a fragments de l'URL. Per exemple en el *framework* Express es pot definir una ruta de la manera següent:

```
1  aplicacio.get('/provincies/:provincia/poblacions/:poblacio', function (peticio,
       resposta) {
2    resposta.send(peticio.params)
3  })
```

Aquest encaminament acceptaria un URL com `/provincies/Barcelona/poblacions/Martorell`, que generaria com a paràmetres l'objecte que podeu veure a continuació. És a dir, tant `:provincia` com `:població` són interpretats com a paràmetres d'encaminament:

```
1  {
2    provincia: Barcelona,
3    poblacio: Martorell
4  }
```

Una característica comuna a tots els sistemes d'encaminament és que també permeten utilitzar expressions regulars, de manera que es poden encaminar les rutes que comencin, acabin o continguin una combinació concreta de caràcters, per exemple.

Els sistemes d'encaminament més avançats també inclouen l'opció d'afegir `middleware`, funcions que són cridades amb la resposta i la petició i que permeten modificar-la abans (o després) de passar-les als controladors. D'aquesta manera es poden afegir sistemes d'autenticació, validacions de dades, adaptació de les dades, etc.

Com es pot comprovar, les implementacions pròpies dels sistemes d'encaminament acostumen a ser molt enrevessades i limitades en comparació amb els mòduls i *frameworks* especialitzats. Per aquesta raó només és recomanable utilitzar-les en els casos més simples.

Lectura i escriptura de fitxers

Les aplicacions desenvolupades amb Node.js, al contrari que les desenvolupades pel navegador, poden llegir i escriure fitxers al disc. Això permet utilitzar sistemes de persistència propis o guardar un registre d'esdeveniments.

Vegeu a continuació un exemple que permet afegir el nom i el cognom d'un usuari a un fitxer que més endavant serà recuperat per poder realitzar cerques i mostrar llistats. A la banda del client només es necessita el codi per realitzar l'enviament utilitzant el mètode `POST` i afegir les dades `nom` i `cognom`:

```
<html>
<body>
  <h1>Petició</h1>
  <label>Nom:<input id="nom" /></label>
  <label>Cognom:<input id="cognom" /></label>
  <button id="afegir_boto">Afegir nom</button>
  <h1>Resposta</h1>
  <div id="resposta"></div>
  <script>
    var afegirBoto = document.getElementById('afegir_boto');
    var resposta = document.getElementById('resposta');
    var metode = "POST"

    var nom = document.getElementById('nom');
    var cognom = document.getElementById('cognom');

    afegirBoto.addEventListener("click", function(e) {
      var url = 'http://127.0.0.1:8080';
      var dades = 'nom=' + encodeURI(nom.value);
        dades += '&cognom=' + encodeURI(cognom.value);

      var httpRequest = new XMLHttpRequest();
      httpRequest.open('POST', url, true);
      httpRequest.send(dades);

      httpRequest.onload = function (e) {
        resposta.innerHTML = httpRequest.responseText;
      }
    });
  </script>
</body>
</html>
```

A la banda del servidor, cal carregar el mòdul `fs` (sigles de *file system*, 'sistema de fitxers' en anglès). Per guardar la informació s'ha optat per utilitzar un fitxer de text pla en el qual el nom i el cognom estaran separats per una barra vertical `|`, mentre que cada parell de `nom` i `cognom` estarà separat per un asterisc `*`.

El mòdul 'FileSystem'

Podeu trobar tota la documentació sobre el mòdul `FileSystem` a l'enllaç següent: goo.gl/tpurw0.

Com que només s'accepten peticions de tipus `POST` no cal cercar la cadena de consulta a l'URL, sempre es trobarà al cos de la petició. Per aquest mateix motiu s'ha restringit l'accés a aquest servei només a les peticions `POST`, i s'estableix la limitació a la capçalera de la resposta.

```
var http = require('http');
var queryString = require('querystring');
var fs = require('fs');

var server = http.createServer();
server.on('request', function(peticio, resposta) {
  resposta.setHeader('Access-Control-Allow-Origin', '*');
  resposta.setHeader('Access-Control-Allow-Methods', 'POST');
  resposta.setHeader('Content-Type', 'text/plain;charset=utf-8');

  var cos = '';
  peticio.on('data', function(dades) {
    if (cos.length > 1e6) {
      peticio.connection.destroy();
    } else {
      cos += dades;
    }

  }).on('end', function() {
    var params = queryString.parse(cos);
    var nom = decodeURI(params.nom);
    var cognom = decodeURI(params.cognom);

    var data = Math.floor(Date.now() / 1000);
    fs.appendFile('noms.txt', nom + "|" + cognom + "*", function(error) {
      var missatge;

      if (error) {
        throw error; // No continuarà l'execució
      }

      resposta.statusCode = 200;
      missatge = 'Dades afegides al fitxer correctament: ' + nom + ' ' + cognom
          ;
      resposta.write(missatge);
      console.log(missatge);
      resposta.end();
    });
  });
});

server.listen(8080, '127.0.0.1');
```

Fixeu-vos que en lloc d'utilitzar `writeHead` per establir el codi d'estat, s'ha modificat directament la propietat, ja que aquest valor variarà segons si s'ha produït o no un error, i en el seu lloc s'ha establert el tipus de contingut utilitzant el mètode `setHeader`.

En aquest cas, no es considera l'opció d'acceptar altres mètodes, així que s'ha restringit l'accés al servei web de manera que només accepta peticions enviades mitjançant el mètode `POST`.

Com es pot apreciar, per afegir nou contingut al fitxer, s'utilitza el mètode `fs.appendFile` passant com a paràmetres el nom del fitxer (en aquest cas `noms.txt`), el contingut que s'ha d'afegir i una funció que es cridarà en acabar d'escriure les dades.

Aquesta funció pot rebre un error com a argument si s'ha produït en desar el fitxer (per exemple, per manca de permisos d'escriptura o si el disc fos ple). Per aquesta raó, es comprova si s'ha rebut un error i es genera el codi de la resposta i un missatge adient que es mostrarà per la consola.

Si proveu d'enviar dades a través del navegador, veureu que a la consola apareixeran els continguts de les variables `nom` i `cognom` que s'han afegit al client, i es crearà el fitxer amb el format especificat: parells de `nom` i `cognom` separats per una barra vertical, i aquests separats d'altres parells per un asterisc.

Cal tenir en compte que el mètode `appendFile` permet afegir dades a un fitxer existent, però en cas que aquest no existeixi en crearà un de nou. Un mètode similar del mòdul `FileSystem`, anomenat `writeFile`, permet escriure dades al disc, però en aquest cas sempre se sobreescriu el fitxer.

Aquesta última opció pot interessar si es treballa amb altre tipus de continguts. Per exemple, si en lloc de desar el fitxer com a text pla el guardem com a JSON, s'haurà de guardar l'objecte complet, que contindrà tota la informació. Això obligarà a llegir el fitxer en arrencar el servidor (si no, s'esborraria tota la informació anterior). Vegeu com es podria implementar en l'exemple següent:

```
var http = require('http');
var queryString = require('querystring');
var fs = require('fs');

var dadesAplicacio;

const NOM_FITXER = 'noms.json';

carregarDadesJSON(NOM_FITXER);

var server = http.createServer();

server.on('request', function(peticio, resposta) {
  resposta.setHeader('Access-Control-Allow-Origin', '*');
  resposta.setHeader('Access-Control-Allow-Methods', 'POST');
  resposta.setHeader('Content-Type', 'text/plain;charset=utf-8');

  var cos = '';
  peticio.on('data', function(dades) {
    if (cos.length > 1e6) {
      peticio.connection.destroy();
    } else {
      cos += dades;
    }

  }).on('end', function() {
    var params = queryString.parse(cos);
    var nom = decodeURI(params.nom);
    var cognom = decodeURI(params.cognom);

    dadesAplicacio[dadesAplicacio.length] = {
      nom: nom,
      cognom: cognom
    };
    desarDadesJSON(NOM_FITXER, dadesAplicacio);
    missatge = 'Afegit ' + nom + ' ' + cognom + ' al fitxer';
    resposta.write(missatge);
    console.log(missatge);
    resposta.end();
  });
});

function desarDadesJSON(fitxer, dades) {
  fs.writeFile(fitxer, JSON.stringify(dades), function(error) {
    if (error) {
      console('Error en desar el fitxer');
      throw err;
    } else {
      console.log('Dades desades');
```

```
    }
  });
}

function carregarDadesJSON(fitxer) {
  fs.readFile(fitxer, 'utf8', function(err, dades) {
    if (err) {
      console.log('No existeix el fitxer, creant nou conjunt de dades');
      dadesAplicacio = [];
    } else {
      console.log('Dades carregades:', dades);
      dadesAplicacio = JSON.parse(dades);
    }
  });
}

server.listen(8080, '127.0.0.1');
```

Fixeu-vos que s'ha definit una variable global per al mòdul anomenada `dadesAplicacio`. Aquesta variable contindrà l'*array* amb les dades. Cal destacar que mentre el servidor continuï funcionant, aquestes dades continuaran en memòria, és a dir, no es perden entre peticions. Per aquesta raó, només cal llegir el fitxer en iniciar el servidor, ja que cada vegada que es rep una nova petició s'afegeix un nou element a l'*array* amb aquesta informació i es desa al fitxer.

Com es pot apreciar, a diferència de JavaScript, Node.js sí que disposa de constants; en aquest cas s'ha definit la constant `NOM_FITXER` amb el nom del fitxer que contindrà les dades.

Cal destacar que la lectura i l'escriptura de dades utilitzant els mètodes `readFile`, `appendFile` i `writeFile` és asíncrona: això vol dir que mentre s'estan realitzant les operacions, el programa continua executant-se. Per aquesta raó no es pot retornar el contingut del fitxer en carregar-lo, ja que el valor de retorn és `undefined` en aquell moment. Per aquesta raó, el que s'ha fet és establir el valor de la variable global `dadesAplicacio` dintre de la funció que és cridada en finalitzar la lectura.

Com que s'ha utilitzat el format JSON només cal fer servir els mètodes `JSON.stringify` per convertir les dades en una cadena de text per desar i `JSON.parse` per convertir la cadena de text en un objecte de JavaScript. Aquest format simplifica molt la manipulació de dades, ja que no cal que implementeu els vostres propis analitzadors i permet treballar directament amb les dades llegides.

1.1.5 Cas pràctic: implementació d'un servei web REST

Una vegada ja se sap com fer l'encaminament dels URL, com es llegeixen i s'escriuen els fitxers i com es generen les respostes, implementar un servei web és relativament simple. Com a exemple s'implementarà un servei web per gestionar esdeveniments que inclourà l'opció de filtratge (podeu trobar el codi a l'enllaç següent: goo.gl/tZMFpu). Al llarg d'aquest exemple es practicaran les competències següents:

- Creació d'un servei web REST amb Node.js que permet realitzar el conjunt d'operacions CRUD (creació, lectura, actualització i eliminació).
  - Generació d'una resposta aplicant mecanismes CORS per permetre l'accés dels mètodes acceptats des de qualsevol domini.
  - Obtenció dels paràmetres de la petició sigui quin sigui el mètode emprat (`POST`, `GET`, `PUT` o `DELETE`).
  - Encaminament de la petició al controlador de recurs adequat segons l'URL.
  - Gestió de les operacions sobre el recurs.
  - Lectura i escriptura de fitxers en format JSON.
- Creació d'una aplicació d'una sola pàgina que permet veure el llistat d'esdeveniments, filtrar els resultats, afegir nous esdeveniments, actualitzar-los i eliminar-los.
  - Utilització de la biblioteca jQuery per simplificar el codi.
  - Utilització de formularis.
  - Utilització d'atributs propis (HTML5).
  - Enviament de peticions AJAX.
  - Manipulació del DOM per crear nous elements i modificar-ne la visibilitat mitjançant CSS.
  - Aplicació del patró mòdul per encapsular l'aplicació.

En anglès, les sigles CRUD es corresponen amb les inicials de *create, read, update, delete*.

Tot i que en aquest exemple només es farà servir un fitxer CSS i un fitxer JavaScript, és recomanable crear una carpeta per a cada tipus, ja que és una bona pràctica a l'hora de desenvolupar aplicacions més complexes.

El contingut del fitxer CSS és força simple; podeu crear un fitxer anomenat `principal.css` amb el codi que trobareu a continuació i desar-lo al directori `css`. A banda de donar estil a la pàgina, s'han afegit les classes `error`, `info`, `exit` i `alerta`, que es fan servir per donar diferents colors a les notificacions segons el seu tipus, i la classe `amaga`, per ocultar els continguts.

```
1  body {
2      width: 700px;
3      margin: 0 auto;
4  }
5
6  h1, h2 {
7      text-align: center;
8  }
9
10 label {
11     display: block;
12     font-weight: bold;
13 }
14
15 input, select, button, textarea {
16     display: block;
17     width: 100%;
18 }
19
20 table {
```

```
    border: 1px solid black;
    width: 100%;
    margin-top: 15px;
    margin-bottom: 15px;
}

table button {
    width: 33%;
    display: inline-block;
    overflow: hidden;
}

.columna-doble {
    display: inline-block;
    width: 49%;
}

.amaga {
    display: none;
}

.buit {
    text-align: center;
    font-style: italic;
}

#notificacions {
    border: 0;
    padding: 5px;
}

#notificacions.error {
    color: #D8000C;
    background-color: #FFBABA;
    border-color: #D8000C;
    border: 1px solid;
}

#notificacions.info {
    color: #00529B;
    background-color: #BDE5F8;
    border-color: #00529B;
    border: 1px solid;
}

#notificacions.exit {
    color: #4f8A10;
    background-color: #DFF2BF;
    border-color: #4f8A10;
    border: 1px solid;
}

#notificacions.alerta {
    color: #9F6000;
    background-color: #FEEFB3;
    border-color: #9F6000;
    border: 1px solid;
}
```

El codi HTML inclou la càrrega del full d'estil, la biblioteca jQuery i el fitxer amb el codi JavaScript. Cal destacar que es divideix en dues parts importants. Per una banda, la capçalera (etiqueta `header`), que inclou el títol de l'aplicació i l'àrea on es mostraran les notificacions i, per altra, la zona principal (etiqueta `main`), que mostrarà els diferents continguts.

```
<!DOCTYPE html>
<html lang="ca">
```

```
<head>
    <meta charset="UTF-8">
    <title>Gestor d'esdeveniments</title>
    <link rel="stylesheet" type="text/css" href="css/principal.css">
</head>
<body>

<header>
    <h1>Gestor d'esdeveniments</h1>
    <div id="notificacions"></div>
</header>

<main>
    <div id="paginaLlistat">
        <h2>Llistat d'esdeveniments</h2>

        <input class="columna-doble" type="text"
               placeholder="Introdueix una paraula o deixa-ho buit per eliminar el filtre"
               name="filtre"/>
        <button class="columna-doble" data-accio="filtrar">Filtrar</button>

        <table>
            <thead>
            <tr>
                <th>Nom</th>
                <th>Data</th>
                <th>Organitzador</th>
                <th>Accions</th>
            </tr>
            </thead>
            <tbody>
            <tr>
                <td colspan="4" class="buit">No s'ha trobat cap esdeveniment</td>
            </tr>
            </tbody>
        </table>
        <button data-accio="mostrar-alta">Afegir nou esdeveniment</button>
    </div>

    <div id="paginaAlta" class="amaga">
        <h2>Alta d'esdeveniment</h2>
        <form id="formulariAlta">
            <label>Nom</label>
            <input type="text" name="nom" placeholder="Nom de l'esdeveniment..." required/>

            <label>Descripció</label>
            <textarea name="descripcio" placeholder="Descripció de l'esdeveniment..."></textarea>

            <label>Data</label>
            <input type="date" name="data"/>

            <label>Organitzador</label>
            <select name="organitzador">
                <option value="Ajuntament de Barcelona" selected>Ajuntament de Barcelona</option>
                <option value="Associació Cultural">Associació Cultural</option>
                <option value="Associació de Botiguers">Associació de Botiguers</option>
                <option value="Organitzador privat">Organitzador privat</option>
            </select>
            <button class="columna-doble" data-accio="alta">Alta</button>
            <button class="columna-doble" data-accio="mostrar-llistat">Tornar al llistat</button>
        </form>
```

```
64      </div>
65
66      <div id="paginaActualitzar" class="amaga">
67          <h2>Modificació d'esdeveniment</h2>
68          <form id="formulariActualitzar">
69              <label>Nom</label>
70              <input type="text" name="nom" placeholder="Nom de l'esdeveniment...
                    " required/>
71
72              <label>Descripció</label>
73              <textarea name="descripcio" placeholder="Descripció de l'
                    esdeveniment..."></textarea>
74
75              <label>Data</label>
76              <input type="date" name="data"/>
77
78              <label>Organitzador</label>
79              <select name="organitzador">
80                  <option selected>Ajuntament de Barcelona</option>
81                  <option>Associació Cultural</option>
82                  <option>Associació de Botiguers</option>
83                  <option>Organitzador privat</option>
84              </select>
85              <button class="columna-doble" data-accio="actualitzar">Actualitzar<
                    /button>
86              <button class="columna-doble" data-accio="mostrar-llistat">Tornar
                    al llistat</button>
87          </form>
88      </div>
89
90      <div id="paginaDetall" class="amaga">
91          <h2>Detall d'esdeveniment</h2>
92          <div>
93              <label>Nom</label>
94              <span id="detallNom">xx</span>
95              <label>Descripció</label>
96              <span id="detallDescripcio">xx</span>
97              <label>Data</label>
98              <span id="detallData">xx</span>
99              <label>Organitzador</label>
100             <span id="detallOrganitzador">xx</span>
101         </div>
102         <button class="columna-doble" data-accio="mostrar-actualitzar">
                Actualitzar</button>
103         <button class="columna-doble" data-accio="mostrar-llistat">Tornar al
                llistat</button>
104     </div>
105 </main>
106
107 <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
108 <script src="js/gestor-esdeveniments.js"></script>
109 </body>
110 </html>
```

Fixeu-vos que la zona principal conté quatre contenidors `div`: `paginaLlistat`, `paginaAlta`, `paginaActualitzar`, `paginaDetall`. Aquests contenidors representen les diferents pantalles de l'aplicació i, inicialment, la classe `amaga` s'aplica a totes excepte a `paginaLlistat`. Mitjançant JavaScript es fa la transició entre pantalles afegint aquesta classe a la pàgina anterior i eliminant-la de la pàgina nova.

Cal destacar que als botons s'ha aplicat l'atribut personalitzat `data-accio` i en alguns casos, mitjançant JavaScript, l'atribut `data-id`. Això permet processar tots els botons amb el mateix detector d'*events* i segons el valor que tinguin aquests atributs es realitzarà una acció o una altra. Per exemple, en fer clic sobre el botó

Detall es processarà l'acció `mostrar-detall` amb l'identificador corresponent a l'esdeveniment (indicat per `data-id`).

Com es pot apreciar, la taula és buida en el codi HTML i les files es generen mitjançant JavaScript una vegada es rep el llistat d'esdeveniments del servidor.

Un detall a tenir en compte és que el tipus de dades `date` no forma part de l'especificació d'HTML5, sinó que es tracta d'una implementació pròpia de Google Chrome i, per tant, no és disponible en altres navegadors (com Mozilla Firefox). En aquests casos es tractarà com un camp de text normal.

El codi del client el podeu guardar en un fitxer anomenat `gestor-esdeveniments` dintre del directori `js`. Com podeu veure a continuació, s'ha aplicat el patró mòdul per encapsular-lo, de manera que tota l'aplicació es troba a la variable `GESTOR_ESDEVENIMENTS` i només és accessible externament el mètode `iniciarAplicació`. D'aquesta manera es pot assegurar que no interferirà amb altres aplicacións ni serà manipulada externament.

S'ha de tenir en compte que a `GESTOR_ESDEVENIMENTS` no s'assigna una funció, sinó que **se n'assigna el retorn**, ja que és una funció autoinvocada i, per consegüent, el contingut de `GESTOR_ESDEVENIMENTS` és un objecte amb un únic mètode (`iniciarAplicació`) que té accés a l'aplicació gràcies a la clausura.

```
1  var GESTOR_ESDEVENIMENTS = (function () {
2
3      var paginaMostradaActualment;
4      var llistatEsdeveniments;
5
6      function enviarPeticio(metode, url, params) {
7          $.ajax({
8              url: url,
9              data: params,
10             dataType: 'json',
11             type: metode,
12
13             error: function () {
14                 accioMostrarNotificacio('error', 'S\'ha produït un error en fer
                       la petició al servidor');
15             },
16
17             success: function (dades) {
18                 processarResposta(dades);
19             }
20         });
21     }
22
23     function processarAccio(accio, dades) {
24         switch (accio) {
25             case 'establir_llistat':
26                 accioEstablirLlistat(dades.esdeveniments);
27                 break;
28
29             case 'mostrar_notificacio':
30                 accioMostrarNotificacio(dades.notificacio.tipus, dades.
                       notificacio.missatge);
31                 break;
32
33             case 'mostrar_pagina':
34                 accioMostrarPagina(dades.pagina);
35                 break;
36
37             case 'mostrar_actualitzar':
```

```
                establirEsdevenimentActualitzar(llistatEsdeveniments[dades.id],
                     'actualitzar');
                accioMostrarPagina('paginaActualitzar');
                break;

            case 'mostrar_detall':
                establirEsdevenimentDetall(llistatEsdeveniments[dades.id], '
                    detall');
                accioMostrarPagina('paginaDetall');
                break;

            case 'mostrar_llistat':
                netejarFormularis();
                accioMostrarPagina('paginaLlistat');
                break;

            case 'mostrar_alta':
                netejarFormularis();
                accioMostrarPagina('paginaAlta');
                break;

            case 'eliminar':
                enviarPeticio('DELETE', 'http://127.0.0.1:8080/esdeveniments/'
                    + dades.id);
                break;

            case 'actualitzar':
                dades.esdeveniment = obtenirEsdevenimentDelFormulari('
                    formulariActualitzar');
                dades.esdeveniment.id = dades.id;
                enviarPeticio('PUT', 'http://127.0.0.1:8080/esdeveniments/' +
                    dades.id, dades.esdeveniment);
                break;

            case 'alta':
                dades.esdeveniment = obtenirEsdevenimentDelFormulari('
                    formulariAlta');
                enviarPeticio('POST', 'http://127.0.0.1:8080/esdeveniments',
                    dades.esdeveniment);
                break;

            case 'filtrar':
                accioFiltrar();
                break;

            default:
                console.error('Acció desconeguda: ', accio);
        }
    }

    function processarResposta(dades) {

        for (var i = 0; i < dades.accions.length; i++) {
            processarAccio(dades.accions[i].accio, dades.accions[i])
        }
    }

    function processarBoto(e) {
        e.preventDefault();

        var $boto = $(this);
        var accio = $boto.attr('data_accio');
        var id = $boto.attr('data_id');
        var dades = {
            $boto: $boto,
            id: id
        };

        processarAccio(accio, dades);
    }
```

```
101
102     function accioMostrarNotificacio(tipus, missatge) {
103         var $notificacions = $('#notificacions');
104         $notificacions.removeClass();
105         $notificacions.addClass(tipus);
106         $notificacions.html(missatge);
107     }
108
109     function accioMostrarPagina(novaPagina) {
110         $('#' + paginaMostradaActualment).addClass('amaga');
111         paginaMostradaActualment = novaPagina;
112         $('#' + paginaMostradaActualment).removeClass('amaga');
113     }
114
115     function accioEstablirLlistat(dades) {
116         llistatEsdeveniments = dades;
117         omplirTaulaLlistat(dades);
118         accioMostrarPagina('paginaLlistat');
119     }
120
121     function accioFiltrar() {
122         var $filtre = $('input[name="filtre"]');
123         var terme = $filtre.val();
124
125         if (terme.indexOf(' ') !== -1) {
126             accioMostrarNotificacio('error', 'Només es pot introduir una
                    paraula per filtrar');
127         } else if (!terme) {
128             accioMostrarNotificacio('info', 'Desactivat el filtre');
129             omplirTaulaLlistat(llistatEsdeveniments);
130         } else {
131             filtrarEsdeveniments(terme);
132             $filtre.val('');
133         }
134     }
135
136     function omplirTaulaLlistat(dades) {
137         var $cosTaula = $('tbody');
138
139         // Es neteja el cos del llistat
140         $cosTaula.empty();
141
142         // Es recorren les dades
143         for (var i in dades) {
144
145             // Es genera la fila i les columnes
146             var $fila = $('<tr>');
147             var $col1 = $('<td>');
148             var $col2 = $('<td>');
149             var $col3 = $('<td>');
150             var $col4 = $('<td>');
151
152
153             // S'afegeix el contingut a cada columna a partir de la informació
                    del llistat de dades
154             $col1.html(dades[i].nom);
155             $col2.html(dades[i].data);
156             $col3.html(dades[i].organitzador);
157
158             var $botoDetall = $('<button>Detall</button>');
159             $botoDetall.attr('data-id', dades[i].id);
160             $botoDetall.attr('data-accio', 'mostrar-detall');
161
162             var $botoActualitzar = $('<button>Actualitzar</button>');
163             $botoActualitzar.attr('data-id', dades[i].id);
164             $botoActualitzar.attr('data-accio', 'mostrar-actualitzar');
165
166             var $botoEliminar = $('<button>Eliminar</button>');
167             $botoEliminar.attr('data-id', dades[i].id);
168             $botoEliminar.attr('data-accio', 'eliminar');
```

```
169
170             $col4.append($botoDetall);
171             $col4.append($botoActualitzar);
172             $col4.append($botoEliminar);
173
174             // S'afegeixen les columnes a la fila
175             $fila.append($col1);
176             $fila.append($col2);
177             $fila.append($col3);
178             $fila.append($col4);
179
180             // S'afegeix la fila al cos de la taula
181             $cosTaula.append($fila);
182         }
183
184         // S'afegeix un detector d'events a tots els botons
185         $('table button').on('click', processarBoto);
186     }
187
188     function filtrarEsdeveniments(terme) {
189         var llistatFiltrat = [];
190
191         for (var i in llistatEsdeveniments) {
192             if (llistatEsdeveniments[i].nom.indexOf(terme) !== -1
193                 || llistatEsdeveniments[i].descripcio.indexOf(terme) !== -1
194                 || llistatEsdeveniments[i].organitzador.indexOf(terme) !== -1
195             ) {
196                 llistatFiltrat.push(llistatEsdeveniments[i]);
197             }
198         }
199
200         omplirTaulaLlistat(llistatFiltrat);
201
202         if (llistatFiltrat.length === 0) {
203             accioMostrarNotificacio('alerta', 'No s\'ha trobat cap resultat');
204         } else {
205             accioMostrarNotificacio('exit', 'Llistat filtrat pel terme "' +
                    terme + '".');
206         }
207     }
208
209     function obtenirEsdevenimentDelFormulari(idFormulari) {
210         var $form = $('form#' + idFormulari);
211
212         return {
213             nom: $form.find('input[name="nom"]').val(),
214             descripcio: $form.find('textarea').val(),
215             data: $form.find('input[name="data"]').val(),
216             organitzador: $form.find('select').val()
217         }
218     }
219
220     function establirEsdevenimentActualitzar(esdeveniment) {
221         var $form = $('form');
222
223         $form.find('input[name="nom"]').val(esdeveniment.nom);
224         $form.find('textarea').val(esdeveniment.descripcio);
225         $form.find('input[name="data"]').val(esdeveniment.data);
226         $form.find('select').val(esdeveniment.organitzador);
227         $form.find('[data-accio="actualitzar"]').attr('data-id', esdeveniment.
                id);
228     }
229
230     function establirEsdevenimentDetall(esdeveniment) {
231         $('#detallNom').html(esdeveniment.nom);
232         $('#detallDescripcio').html(esdeveniment.descripcio);
233         $('#detallData').html(esdeveniment.data);
234         $('#detallOrganitzador').html(esdeveniment.organitzador);
235
```

```
236         $('[data-accio="mostrar-actualitzar"]').attr('data-id', esdeveniment.id
                );
237     }
238
239     function netejarFormularis() {
240         $('form').trigger('reset');
241     }
242
243     return {
244         iniciarAplicacio: function () {
245             $('button').on('click', processarBoto);
246             enviarPeticio('GET', 'http://127.0.0.1:8080/esdeveniments');
247         }
248     }
249
250 })();
251
252 $(document).ready(function () {
253     GESTOR_ESDEVENIMENTS.iniciarAplicacio();
254 });
```

Fixeu-vos que s'ha centralitzat el processament d'accions en la funció `processarAccio`, que rep el nom de l'acció que cal portar a terme i un objecte amb les dades que s'han de processar. Aquesta funció és cridada tant pel processament de les respostes que arriben del servidor com pel processament dels botons, de manera que aquestes funcions només s'han d'encarregar d'enviar les dades correctes per realitzar el processament.

Les accions en conjunt es poden dividir en dos grups: les responsables d'enviar peticions al servidor i les responsables de modificar-ne la vista (la representació de les dades a la pantalla). Cal tenir en compte que, per simplificar, no s'ha creat una acció per a cada cas, ja que moltes només requereixen un parell de línies de codi. En projectes més complexos, però, és una bona pràctica (o fins i tot utilitzar diferents objectes).

La funció `processarResposta` és l'encarregada de processar les respostes a les peticions AJAX. Aquestes respostes han de ser en format JSON i han de contenir un *array* amb el nom `accions`, que contindrà una o més accions per processar. D'aquesta manera una resposta pot contenir múltiples accions, com poden ser afegir una notificació, actualitzar el llistat i mostrar una pàgina diferent, i són portades a terme per la funció `processarAcció`.

Per simplificar l'enviament de peticions AJAX s'ha creat una funció anomenada `enviarPeticio`, que és l'encarregada de crear-la, ja que el codi d'aquestes peticions és idèntic en tots els casos i només canvia el mètode d'enviament, l'URL i els paràmetres. Si es produeix cap error, es mostrarà una notificació i en cas d'èxit es processarà la resposta invocant la funció `processarResposta`.

Per altra banda, la funció `processarBoto` és invocada quan es fa clic sobre qualsevol dels botons, tant els inclosos en el codi HTML com els creats dinàmicament en generar la taula. Aquesta funció s'encarrega d'obtenir la informació necessària i passar-la a la funció `processarAcció`.

Les funcions `accioMostrarNotificacio` i `accioMostrarPagina` s'encarreguen de modificar la vista. La primera reemplaça l'estil i contingut de la notificació activa amb la nova, de manera que serà de color verd en cas d'èxit, blau en cas de ser una notificació informativa, groc en cas d'alerta i vermell en

cas d'error. La segona amaga la pàgina anterior i en mostra la nova, afegint i eliminant la classe CSS `amaga`, respectivament.

Les funcions `accioFiltrar` i `filtrarEsdeveniments` són les encarregades de filtrar els resultats. La primera comprova que només s'hagi utilitzat una paraula i discrimina entre aplicar el filtratge o eliminar el filtre (omplint la taula amb tots els esdeveniments). La segona és l'encarregada de realitzar l'acció, recorrent tots els esdeveniments i creant una llista amb els que continguin el terme a les propietats `nom`, `descripcio` o `organitzador`.

Per gestionar els formularis i recopilar-ne les dades es fan servir les funcions `obtenirEsdevenimentDelFormulari`, `establirEsdevenimentActualitzar` i `netejarFormularis`. El primer es fa servir tant en actualitzar un esdeveniment com en donar-ne d'alta un de nou i serveix per obtenir un esdeveniment a partir de les dades del formulari. El segon es fa servir per establir les dades de l'esdeveniment per actualitzar al formulari d'actualització. Finalment, el tercer és una funció auxiliar utilitzada per netejar tota la informació de tots els formularis de la pàgina.

La funció `omplirTaulaLlistat` és l'encarregada de generar la taula amb la informació resumida dels esdeveniments. El primer pas és buidar-la utilitzant el mètode `empty` de jQuery; seguidament es recorren les dades (que corresponen a un objecte contenidor d'esdeveniments), es crea una fila amb 4 columnes i s'afegeix la informació de l'esdeveniment a les 3 primeres i els botons d'acció a l'última.

Una vegada s'ha generat la taula, s'afegeix un detector d'*events* a tots els botons de la taula per invocar el mètode `processarBoto` en detectar-se l'*event* `click`.

Cal tenir en compte un detall important. Fixeu-vos que el llistat d'esdeveniments i les dades no és un *array*, sinó un objecte de JavaScript. Aquesta decisió s'ha pres per simplificar la gestió d'esdeveniments, perquè una vegada s'elimina un element de l'*array*, l'índex dels elements ja no es correspon amb el seu identificador (que s'ha decidit fer seqüencial, començant pel 0). D'aquesta manera, l'identificador de l'esdeveniment s'utilitza com a nom de la propietat a l'objecte i l'eliminació no l'afecta perquè es guarda sempre la referència del pròxim identificador a assignat (gestionat pel servidor).

És a dir, l'estructura de dades que es guarda al servidor és similar a la següent (en format JSON):

```
{
  "proximIdentificador":9,
  "llistatEsdeveniments":{
    "5":{
      "nom":"Primer esdeveniment",
      "descripcio":"Aquesta es la descripció del primer esdeveniment.",
      "data":"2016-12-15",
      "organitzador":"Associació de Botiguers",
      "id":5
    },
    "8":{
      "nom":"Segon esdeveniment",
      "descripcio":"Aquest esdeveniment es realitzarà en una data posterior al
          primer",
      "data":"2017-12-20",
      "organitzador":"Ajuntament de Barcelona",
      "id":8}
```

```
  }
}
```

A continuació podeu trobar el codi del servidor, que fa servir els mòduls de Node.js `http` per crear el servidor, `url` per tractar els URL, `querystring` per gestionar els paràmetres i `fs` per treballar amb el sistema de fitxers.

```
var http = require('http');
var url = require('url');
var queryString = require('querystring');
var fs = require('fs');

const NOM_FITXER = 'esdeveniments.json';
const INDEX_RECURS = 0;
const INDEX_IDENTIFICADOR = 1;

var server = http.createServer();
var dadesAplicacio;

inicialitzar();

function inicialitzar() {
    carregarDadesJSON(NOM_FITXER);

    server.on('request', function (peticio, resposta) {
        resposta.setHeader('Access-Control-Allow-Origin', '*');
        resposta.setHeader('Access-Control-Allow-Methods', 'GET, PUT, POST,
            DELETE');
        resposta.writeHead(200, {'Content-Type': 'application/json;charset=utf
            -8'});

        var cos = '';
        peticio.on('data', function (dades) {
            cos += dades;

        }).on('end', function () {
            var params;
            var dades;

            if (peticio.method === 'GET') {
                dades = url.parse(peticio.url).query;
            } else {
                dades = cos;
            }
            params = queryString.parse(dades);

            var urlEncaminament = (url.parse(peticio.url).path).substr(1);
            encaminar(urlEncaminament, peticio.method, resposta, params);
            resposta.end();

        });
    });

    server.listen(8080, "127.0.0.1");
}

function encaminar(url, metode, resposta, params) {
    var contingutResposta = {};
    var fragmentsUrl = url.split('/');

    switch (fragmentsUrl[INDEX_RECURS]) {
        case 'esdeveniments':
            contingutResposta.accions = controladorEsdeveniments(metode,
                fragmentsUrl[INDEX_IDENTIFICADOR], params);
            break;

        default:
            contingutResposta.accions = [];
            afegirNotificacio(contingutResposta.accions, 'error', 'No es
                reconeix el recurs ' + fragmentsUrl[INDEX_RECURS]);
```

```
60      }
61
62      resposta.write(JSON.stringify(contingutResposta));
63  }
64
65  function controladorEsdeveniments(metode, id, params) {
66      var accions = [];
67
68      switch (metode) {
69          case 'GET':
70              afegirNotificacio(accions, 'info', 'Carregat llistat d\'
                    esdeveniments.');f
71              afegirLlistat(accions);
72              break;
73
74          case 'POST':
75              params.id = dadesAplicacio.proximIdentificador;
76              dadesAplicacio.llistatEsdeveniments[dadesAplicacio.
                    proximIdentificador++] = params;
77
78              desarDadesJSON('esdeveniments.json', dadesAplicacio);
79              afegirNotificacio(accions, 'exit', 'Nou esdeveniment afegit.');
80              afegirLlistat(accions);
81              break;
82
83          case 'DELETE':
84              delete(dadesAplicacio.llistatEsdeveniments[id]);
85              desarDadesJSON('esdeveniments.json', dadesAplicacio);
86              afegirNotificacio(accions, 'info', 'Esdeveniment eliminat.');
87              afegirLlistat(accions);
88              break;
89
90          case 'PUT':
91              dadesAplicacio.llistatEsdeveniments[id] = params;
92              desarDadesJSON('esdeveniments.json', dadesAplicacio);
93              afegirNotificacio(accions, 'exit', 'Esdeveniment actualitzat.');
94              afegirLlistat(accions);
95              break;
96
97          default:
98              afegirNotificacio(accions, 'error', 'No es reconeix el mètode ' +
                    metode);
99      }
100
101     return accions;
102 }
103
104 function afegirNotificacio(contingutResposta, tipus, missatge) {
105     contingutResposta.push({
106         accio: 'mostrar_notificacio',
107         notificacio: {
108             tipus: tipus,
109             missatge: missatge
110         }
111     })
112 }
113
114 function afegirLlistat(contingutResposta) {
115     contingutResposta.push({
116         accio: 'establir_llistat',
117         esdeveniments: dadesAplicacio.llistatEsdeveniments
118     });
119 }
120
121
122 function desarDadesJSON(fitxer, dades) {
123     fs.writeFile(fitxer, JSON.stringify(dades), function (error) {
124         if (error) {
125             console.error('Error en desar el fitxer');
126             throw err;
```

```
127         }
128     });
129 }
130 
131 function carregarDadesJSON(fitxer) {
132     fs.readFile(fitxer, 'utf8', function (err, dades) {
133         if (err) {
134             console.log('No existeix el fitxer de dades, creant nou conjunt de
                    dades...');
135             dadesAplicacio = {
136                 proximIdentificador: 0,
137                 llistatEsdeveniments: {}
138             };
139         } else {
140             dadesAplicacio = JSON.parse(dades);
141         }
142     });
143 }
```

Cal recordar que a Node.js, al contrari que a JavaScript, el concepte de constant sí que existeix i, per consegüent, s'ha utilitzat per definir el nom del fitxer de dades i la correspondència dels índexs en dividir l'URL (el primer correspondrà al tipus de recurs i el segon, si n'hi ha, a l'identificador).

Una vegada s'han importat els mòduls necessaris i s'han definit les constants i variables globals del mòdul, s'invoca la funció `inicialitzar`. Aquesta funció és l'encarregada de carregar les dades i afegir el detector d'*events*, que es dispararà quan el servidor rebi una petició (*event* `request`).

Quan es detecta la petició, la funció executada és l'encarregada d'analitzar l'URL i els paràmetres, i de passar-los a l'encaminador, definit com la funció `encaminar`, que escriurà una resposta o una altra segons les dades de la petició.

Dintre d'aquesta funció, l'URL es divideix en fragments. El primer fragment (amb índex 0) correspon al nom del recurs i el segon fragment (si n'hi ha) correspon a l'identificador. En aquest exemple només es treballa amb un tipus de recurs i, per tant, només hi ha un cas vàlid.

Per generar la resposta, s'ha decidit utilitzar un objecte amb una propietat anomenada *accions*, que contindrà un *array* d'accions per realitzar en el client. Aquestes accions són generades i retornades pel controlador, anomenat `controladorEsdeveniments`, i seran determinades a partir dels paràmetres passats (mètode, identificador i paràmetres).

La funció `controladorEsdeveniments` sempre retornarà un *array* d'accions que, com a mínim, contindrà una notificació d'error. A banda de la resposta, segons el mètode emprat es realitzaran diferents accions sobre les dades:

- **GET**: cap acció al servidor, es retorna el llistat complet i una notificació.
- **POST**: s'afegeix l'identificador a les dades rebudes com a paràmetre i s'afegeix una nova notificació (incrementant posteriorment el valor de `proximIdentificador`). Seguidament es desen les dades en format JSON, s'afegeix una notificació d'èxit i es retorna el llistat.
- **DELETE**: s'elimina l'esdeveniment del conjunt de dades amb l'identificador passat com a paràmetre, a continuació es desen les dades, s'afegeix una

notificació informativa i es retorna el nou llistat (que no inclou l'element eliminat).

- **PUT**: actualitza el conjunt de dades amb la informació de l'esdeveniment, desa les dades i finalment afegeix una notificació.

Fixeu-vos que s'ha fet servir `PUT` per a l'actualització, ja que les dades són completament reemplaçades. En canvi, si l'actualització fos només de determinats valors, llavors seria més apropiat utilitzar el mètode `PATCH`.

Cal destacar que els mètodes `afegirNotificacio` i `afegirLlistat` no retornen res, manipulen directament l'*array* passat com a argument: com que es tracta d'un *array*, es passa per referència i no pas per valor. És a dir, qualsevol canvi realitzat a l'*array* dintre de les funcions `afegirNotificacio` i `afegirLlistat` afecta l'*array* original.

Les funcions `afegirNotificacio` i `afegirLlistat` s'encarreguen de generar correctament l'acció pertinent i d'afegir-la al contingut de la resposta. La primera funció afegeix el tipus de missatge (que afectarà l'estil CSS a aplicar) i el missatge a mostrar, mentre que la segona afegeix el llistat complet d'esdeveniments.

Per acabar, hi ha les funcions `desarDadesJSON` i `carregarDadesJSON`, que són les responsables de carregar el fitxer de dades i desar-lo. En cas de detectar un error en carregar les dades, es genera una nova estructura de dades que serà desada quan es rebi una nova alta d'esdeveniments. Per exemple, en cas de produir-se un error la primera vegada que s'inicia el servidor, es retornarà una estructura de dades buida, que serà desada quan es rebi una petició `POST`; en cas contrari, retornarà l'estructura de dades carregada.

Recordeu que les invocacions a `fs.writeFile` i `fse.readFile` són asíncrones i, per tant, no es bloqueja l'execució mentre es carreguen o es desen les dades (tot i que si és necessari, hi ha una versió síncrona de totes dues).

1.1.6 Cas pràctic: creació d'una aplicació de xat amb sòcols

Aquest exemple mostra com es pot crear una aplicació de xat utilitzant els *frameworks* Express i Socket.io de Node.js (està basat en un dels exemples de la documentació oficial de Socket.io). Podeu trobar el codi complet a l'enllaç següent: goo.gl/jWh4Sb.

El *framework* Express és utilitzat per simplificar les operacions habituals d'un servidor web, com és l'encaminament i l'enviament de fitxers; mentre que Socket.io s'encarrega de la gestió dels sòcols de connexió tant al servidor com al client.

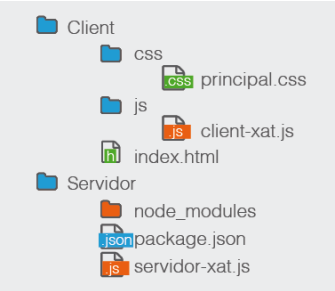
La biblioteca Socket.io utilitza internament l'API WebSockets a la banda del client per realitzar aquestes connexions, facilitant la utilització d'un sistema d'*events* propis que són enviats i gestionats pels sòcols.

En aquest exemple es tractaran els punts següents:

- Instal·lació de *frameworks* de tercers amb Node.js (Express i Socket.io).
- Utilització de *frameworks* per simplificar l'encaminament (Express).
- Encaminament de la petició de fitxers (Express).
- Creació d'un servidor de connexions de sòcols (Socket.io).
- Enviament i detecció d'*events* mitjançant sòcols (Socket.io).

Per reproduir aquest exemple en el vostre entorn, cal crear l'estructura de directoris que es pot veure a la figura 1.8, ja que s'utilitzarà Express per servir els fitxers.

FIGURA 1.8. Estructura de fitxers del xat amb Socket.io



El directori `node_modules` es crearà automàticament en instal·lar els mòduls necessaris utilitzant el gestor de paquets npm.

El primer pas és crear un fitxer anomenat `package.json` amb el següent contingut dintre del directori `Servidor`, que contindrà el servidor de xat:

```
{
  "name": "exemple-web-sockets",
  "version": "0.0.1",
  "description": "Exemple de Xat amb Express i Socket.io",
  "dependencies": {}
}
```

Seguidament, per instal·lar el *framework* Express utilitzant el gestor de paquets npm executeu la següent instrucció:

```
npm install --save express
```

L'opció `-save` fa que la dependència s'afegeixi automàticament al fitxer `package.json` de l'aplicació.

La integració amb el *framework* `Socket.IO` s'ha de fer tant en el client (afegint la biblioteca `socket.io-client`) com en el servidor (carregant el mòdul `socket.io`).

Per afegir el mòdul al servidor s'ha d'executar la comanda següent a la terminal:

```
npm install --save socket.io
```

Una vegada instal·lats els *frameworks* Express i Socket.io, es pot crear dintre de la mateixa carpeta `Servidor` el fitxer `servidor-xat.js`, que contindrà la implementació del servidor de xat.

```
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);
var path = require('path');

// Encaminament
app.get('/', function(peticio, resposta){
    resposta.sendFile(path.resolve(__dirname + '/../Client/index.html'));
});

app.get('/css/:fitxer', function(peticio, resposta){
    resposta.sendFile(path.resolve(__dirname + '/../Client/css/'+ peticio.
        params.fitxer));
});

app.get('/js/:fitxer', function(peticio, resposta){
    resposta.sendFile(path.resolve(__dirname + '/../Client/js/' + peticio.
        params.fitxer));
});

// Gestió de les connexions
io.on('connection', function(socol){
    socol.broadcast.emit('missatge xat', 'Un nou usuari s\'ha connectat');

    console.log('Un usuari connectat');

    socol.on('disconnect', function(){
        socol.broadcast.emit('missatge xat', 'Un usuari s\'ha desconnectat');
        console.log('usuari desconnectat');
    });

    socol.on('missatge xat', function(msg){
        console.log('missatge: ' + msg);
        io.emit('missatge xat', msg);

    });
});

// S'inicia l'escolta del servidor pel port 3000
http.listen(3000, function(){
    console.log('Escoltant a *:3000');
});
```

Primerament, cal importar els mòduls necessaris per a l'aplicació: `express`, `http`, `socket.io` i `path`. Fixeu-vos que en el cas d'`express` i `io` no es carrega el mòdul, sinó que s'invoca com una funció, sense passar cap paràmetre en el cas d'`express` i passant el servidor `http` en el cas de `socket.io`.

Com que `express` no permet utilitzar directament els dos punts per accedir a un directori anterior perquè el considera codi maliciós, per construir la ruta s'ha d'utilitzar el mòdul `path`.

El mòdul `Routing` conté la implementació de l'encaminament d'Express adaptada per funcionar amb el mòdul `Http`.

A continuació es pot veure com funciona l'encaminament d'`Express`, molt més simple que les implementacions manuals. A partir del mètode `get` es passen com a arguments una ruta i la funció que serà cridada en detectar-la.

Es pot trobar més informació sobre els encaminaments d'Express a l'enllaç següent: goo.gl/pgXQuq.

Aquests encaminaments també faciliten treballar amb paràmetres de ruta, prefixats amb el símbol dels dos punts (`:`), als quals es pot accedir com a propietats a partir de l'objecte `peticio.params`. És a dir, quan es demana el fitxer corres-

ponent a la ruta `/js/client-xat.js` el valor de `peticio.params.fitxer` és `client-xat.js`.

Les respostes d'Express ofereixen la possibilitat d'enviar un fitxer directament com a resposta utilitzant el mètode `sendFile`, d'aquesta manera se serveix el fitxer HTML, el fitxer JavaScript i el fitxer CSS corresponent.

Com es pot apreciar, Express reemplaça completament la funcionalitat del mòdul `http`, i fa que la implementació sigui molt més simple i entenedora.

Seguidament es pot veure la implementació de la gestió de connexions. Primerament s'afegeix un detector per a l'*event* `connection` per detectar quan s'ha establert una nova connexió mitjançant l'*event* `connection` del *framework* Socket.io.

Una vegada s'ha rebut una connexió es crida una funció (que rep com a paràmetre el sòcol) perquè realitzi les acions següents:

- S'envia el missatge `Un nou usuari s'ha connectat` a tots els usuaris, excepte al que ha connectat, mitjançant el mètode `socol.broadcast.emit`.
- S'afegeix al sòcol un detector per a l'event `disconnect`, que avisarà els altres usuaris quan un usuari s'hagi desconnectat.
- S'afegeix un detector per a l'*event* `missatge xat` (un *event* propi) que envia a tots els usuaris connectats al servidor (emissor inclòs) el missatge rebut.
- Per facilitar la depuració s'ha afegit un missatge a la consola que permet determinar el moment en què un usuari s'uneix o abandona la sala de xat.

Cal destacar que l'enviament de missatges es tracta, en realitat, d'una emissió d'*events* que farà que l'*event* `missatge xat` es dispari al client amb el missatge enviat com a dades.

Fixeu-vos en les diferències dels dos tipus d'enviament mostrat. Per una banda, hi ha missatges globals que són rebuts per tots els usuaris, i emesos pel servidor (`io.emit`) i, per altra, hi ha els *events* que no ha de rebre l'emissor (com la connexió i la desconnexió) i que són emesos a partir del sòcol (socol.broadcast.emit).

Port de connexió dels sòcols

El port que s'utilitza per connectar amb l'aplicació mitjançant sòcols és el mateix port pel qual s'escolten les peticions HTTP. Així doncs, si es vol canviar aquest port, s'haurà de canviar al servidor modificant la invocació al mètode `http.listen`.

Cal tenir en compte que, per defecte, el client de Socket.io farà servir el mateix port utilitzat per obrir la pàgina, de manera que si la pàgina s'ha obert al port 3000, la connexió dels sòcols es realitzarà a través d'aquest port.

El codi HTML del client ha d'anar en un fitxer anomenat `index.html` dins del directori `Client` amb el contingut següent:

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Client Xat amb Socket.io</title>
    <link rel="stylesheet" type="text/css" href="css/principal.css">
</head>
<body>
<h1>Client Xat</h1>
<ul id="missatges"></ul>
<form action="">
    <input id="missatge" autocomplete="off" /><button>Enviar</button>
</form>

<script src="/socket.io/socket.io.js"></script>
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script src="/js/client-xat.js"></script>

</body>
</html>
```

Com es pot apreciar, només es tracta d'una llista en què es mostraran els missatges, un formulari per entrar el text per enviar al servidor i la càrrega dels fitxers amb el codi JavaScript i les biblioteques necessàries.

Cal destacar que el fitxer `socket.io.js` no es troba realment al servidor, sinó que és generat pel *framework* i, per consegüent, sempre serà servit utilitzant la ruta `/socket.io/socket.io.js`.

Dintre del directori `css` heu de crear un fitxer amb el nom `principal.css` i el codi següent:

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: Arial, "Helvetica Neue", Helvetica, sans-serif
}

h1 {
    text-align: center;
}

form {
    padding: 3px;
    position: fixed;
    bottom: 0;
    width: 100%;
}

form, h1 {
    background: #ccc;
}

form, ul {
    border-top: 1px solid black;
}

form input {
    border: 1px solid black;
    border-radius: 5px;
    padding: 10px;
```

```
    width: 90%;
    margin-right: .5%;
}

form button {
    width: 9%;
    font-weight: bold;
    background: #1b95e0;
    border-radius: 5px;
    border: 1px solid black;
    padding: 10px;
}

#missatges {
    font-family: "Courier New", Courier, "Lucida Sans Typewriter", "Lucida
        Typewriter", monospace    list-style-type: none;
    margin: 0;
    padding: 0;

}

#missatges li {
    padding: 5px;
}

#missatges li:nth-child(odd) {
    background: #eee;
}

.estat {
    color:blue;
}
```

En aquest cas, el codi CSS compleix una funció estrictament decorativa, ja que la funcionalitat de l'aplicació seria la mateixa si no s'hi apliqués cap estil.

Finalment, creeu un fitxer anomenat `client-xat.js` dintre del directori `js` amb el contingut següent per habilitar la connexió del client al servidor:

```
var socol = io();

$('form').submit(function(){
    socol.emit('missatge xat', $('#missatge').val());
    $('#missatge').val('');
    return false;
});

socol.on('missatge xat', function(msg){
    $('#missatges').append($('<li>').text(msg));
});
```

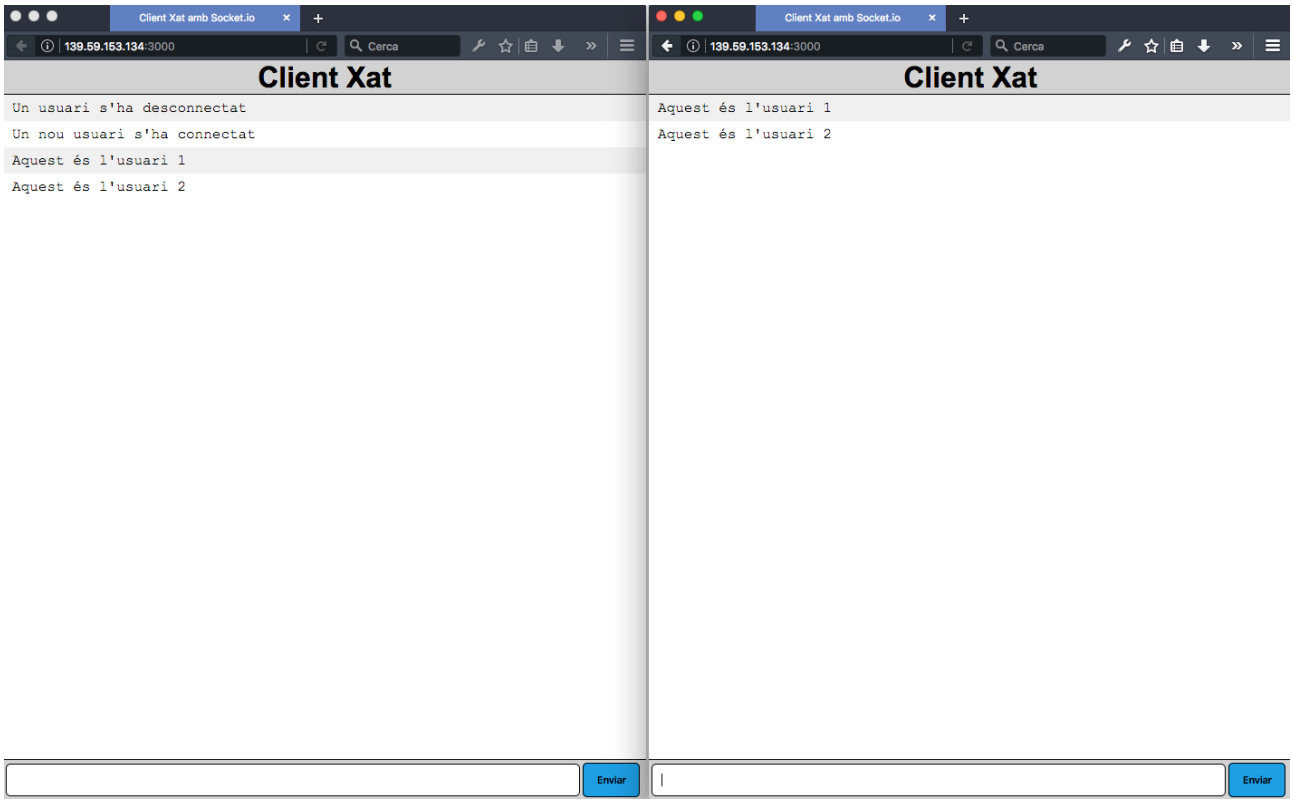
Per iniciar la connexió amb el servidor a través d'un sòcol només cal invocar la funció `io` (o `io.connect`), que pertany a la biblioteca `socket.io.js`. En cas que el client i el servidor es trobessin en diferents dominis o ports caldria especificar l'URL i el port com a paràmetres, però en aquest cas no és necessari.

Una vegada creat el sòcol s'afegeix un detector per a l'*event* `submit` del formulari que detectarà quan s'ha premut el botó d'enviament i, en lloc d'enviar el formulari, emetrà un *event* de tipus `missatge xat` amb el contingut del quadre de text amb l'identificador `missatge` mitjançant el sòcol (que serà rebut pel servidor). Seguidament s'establirà el contingut d'aquest quadre com a buit, a l'espera que s'introdueixi un text nou.

Per altra banda, s'afegeix un detector per a l'*event* `missatge xat` al sòcol, de manera que quan es rep un *event* d'aquest tipus s'executa la funció que s'ha passat com a paràmetre, afegint un nou element de tipus `li` al llistat `missatges` amb el contingut de l'*event*.

A la figura 1.9 es mostra l'aspecte del client funcionant en dues pestanyes diferents.

FIGURA 1.9. Client xat amb dues pestanyes



Fixeu-vos que és molt fàcil crear un protocol propi de comunicació per utilitzar en aplicacions més complexes. Per exemple, es podria discriminar entre els missatges d'estat del sistema i els missatges dels usuaris fent que l'*event* enviat fos diferent, i utilitzar un altre detector per gestionar-los.

Per comprovar-ho substituïu el contingut de la gestió de connexions del fitxer `servidor-xat.js` pel següent:

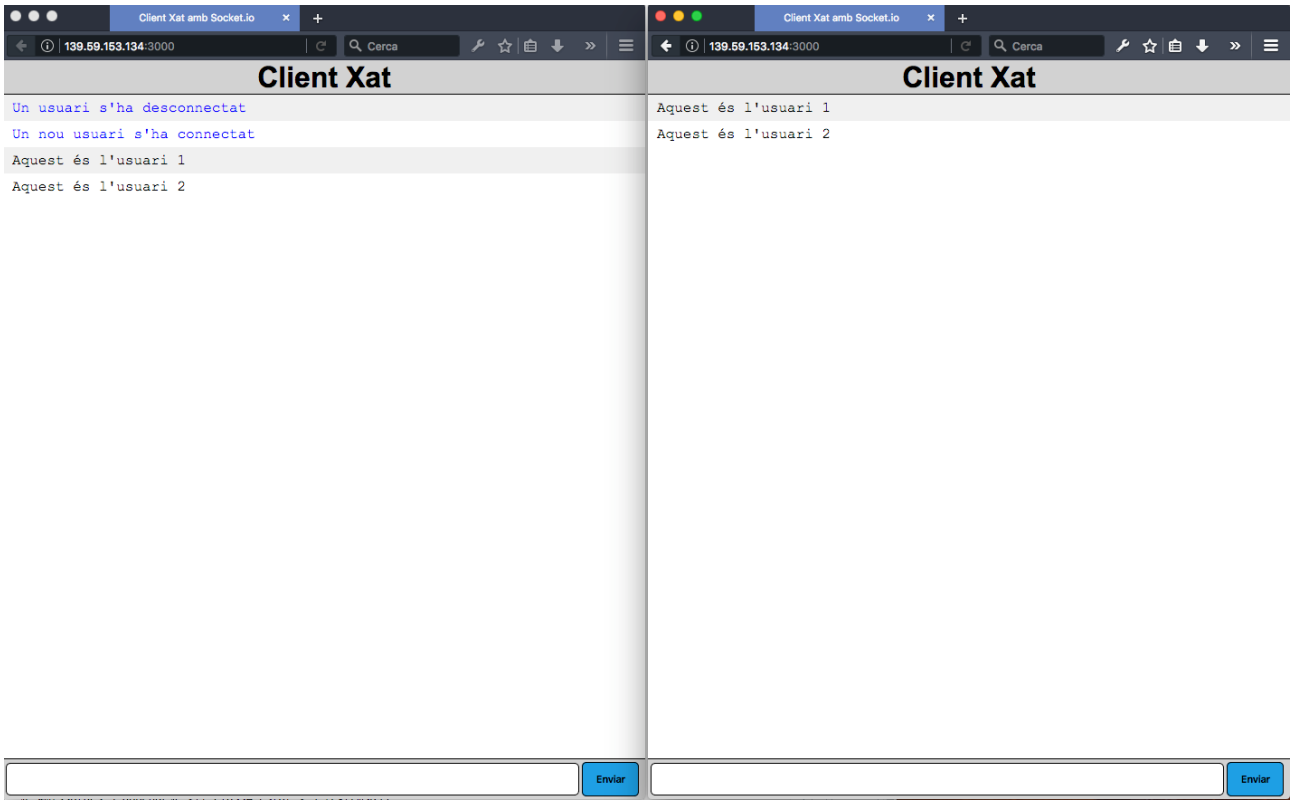
```
// Gestió de les connexions
io.on('connection', function(socol){
    socol.broadcast.emit('missatge estat', 'Un nou usuari s\'ha connectat');

    console.log('Un usuari connectat');

    socol.on('disconnect', function(){
        socol.broadcast.emit('missatge estat', 'Un usuari s\'ha desconnectat');
        console.log('usuari desconnectat');
    });

    socol.on('missatge xat', function(msg){
        console.log('missatge: ' + msg);
        io.emit('missatge xat', msg);

    });
});
```

I afegiu el següent detector el contingut del fitxer `client-xat.js`:

```
1 socol.on('missatge estat', function(msg){
2     $('#missatges').append($('<li class="estat">').text(msg));
3 });
```

Amb aquest canvi, quan es rep un *event* de tipus `missatge estat` s'afegeix un element de tipus `li` amb la classe `estat`, que fa que el missatge es mostri de color blau, tal com podeu veure a la figura 1.10.

FIGURA 1.10. Client xat ampliat amb missatges d'estat



Utilitzant aquest sistema, es podria ampliar l'aplicació per gestionar un llistat d'usuaris o distingir entre missatges globals i missatges privats entre usuaris, per exemple.

Com s'ha pogut comprovar, utilitzant els *frameworks* Express i Socket.io és molt senzill implementar aplicacions, tant a la banda del client com del servidor, que utilitzin sòcols per mantenir una connexió oberta.

1.2 Aplicacions amb la biblioteca Google Maps

Google ofereix moltes API per accedir als seus serveis. La majoria són gratuïtes, però tot i així es necessita crear una clau que permet identificar quines aplicacions estan fent servir cadascun dels serveis.

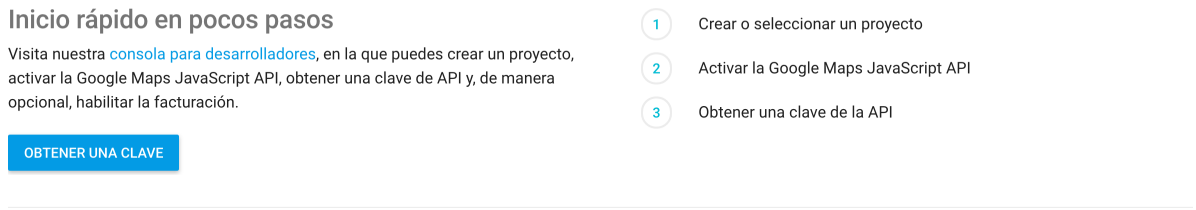
Algunes d'aquestes API, per exemple la de traducció, es poden fer servir fins que s'arriba a uns límits diaris predeterminats. D'altres permeten fer un determinat

nombre de consultes de forma gratuïta i se'n poden contractar més o sol·licitar-ne gratuïtament per a alguns serveis en fase de proves. Aquest nombre d'usos és conegut com a quota.

1.2.1 Obtenció d'una clau per utilitzar l'API Google Maps

En el cas de l'API Google Maps, tot i que per als usos més habituals és gratuïta, és necessari obtenir una clau. Per generar-la s'ha de fer des del panell de desenvolupador de Google, al qual es pot accedir seguint els passos des d'enllaç goo.gl/NnOJk4, i fent clic al botó ***Obtenir una clau***, com es pot veure a la figura 1.11.

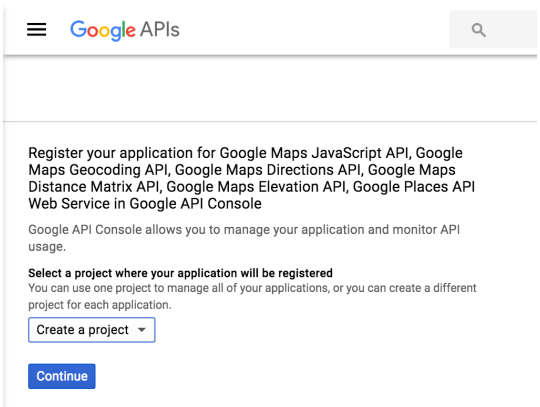
Figura 1.11. Obtenció de la clau per a l'API Google Maps



En cas de no estar autenticat amb un compte de Google, abans de continuar us demanarà les dades de connexió, ja que les vostres aplicacions, les claus i la consola de desenvolupador quedaran lligades al vostre compte.

El següent pas serà crear una aplicació (o seleccionar-ne una d'existent) a la qual s'afegirà la clau, com es mostra a la figura 1.12.

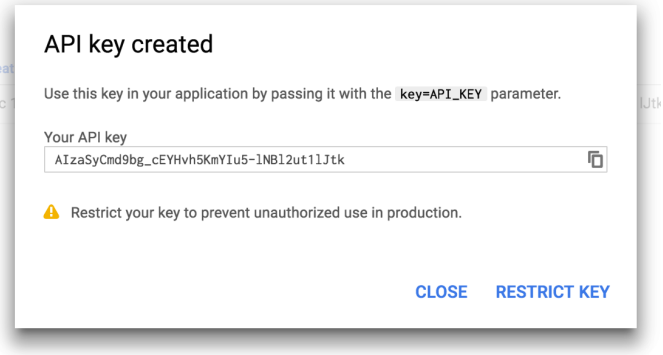
Figura 1.12. Creació d'un nou projecte



Cal tenir en compte que la creació del projecte pot trigar uns quants minuts i en finalitzar es mostrarà el taulell principal de la vostra aplicació automàticament.

A continuació, heu de procedir a generar la clau per a l'aplicació clicant sobre el botó ***Generar clau*** (opcionalment podeu canviar el nom que es mostra per defecte). La nova clau es mostrarà en un diàleg com el que es mostra a la figura 1.13, que podeu procedir a tancar.

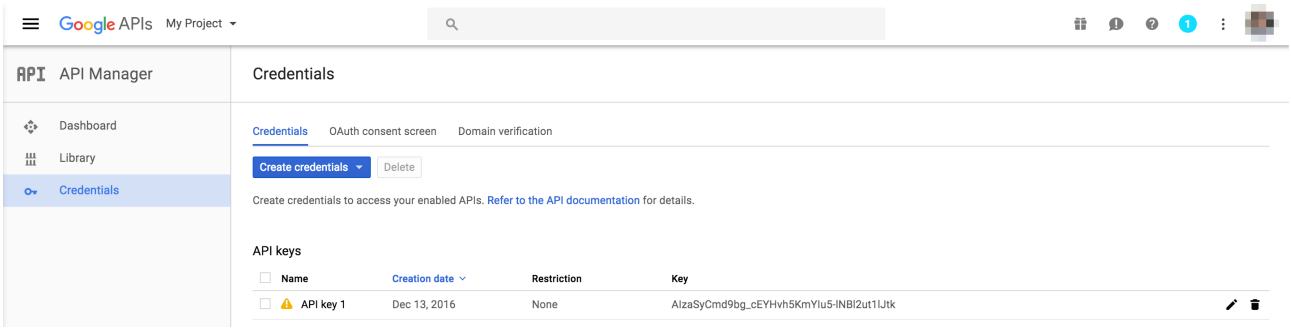
FIGURA 1.13. Diàleg que mostra la clau generada



Com es pot apreciar en el diàleg, us ofereixen l'opció de restringir aquesta clau perquè només sigui utilitzada des de determinats dominis o entorns d'execució. En aquest exemple no s'aplicarà cap restricció, però és un factor que s'ha de tenir en compte per evitar abusos de tercers que facin servir la vostra clau (recordeu que està lligada al vostre compte).

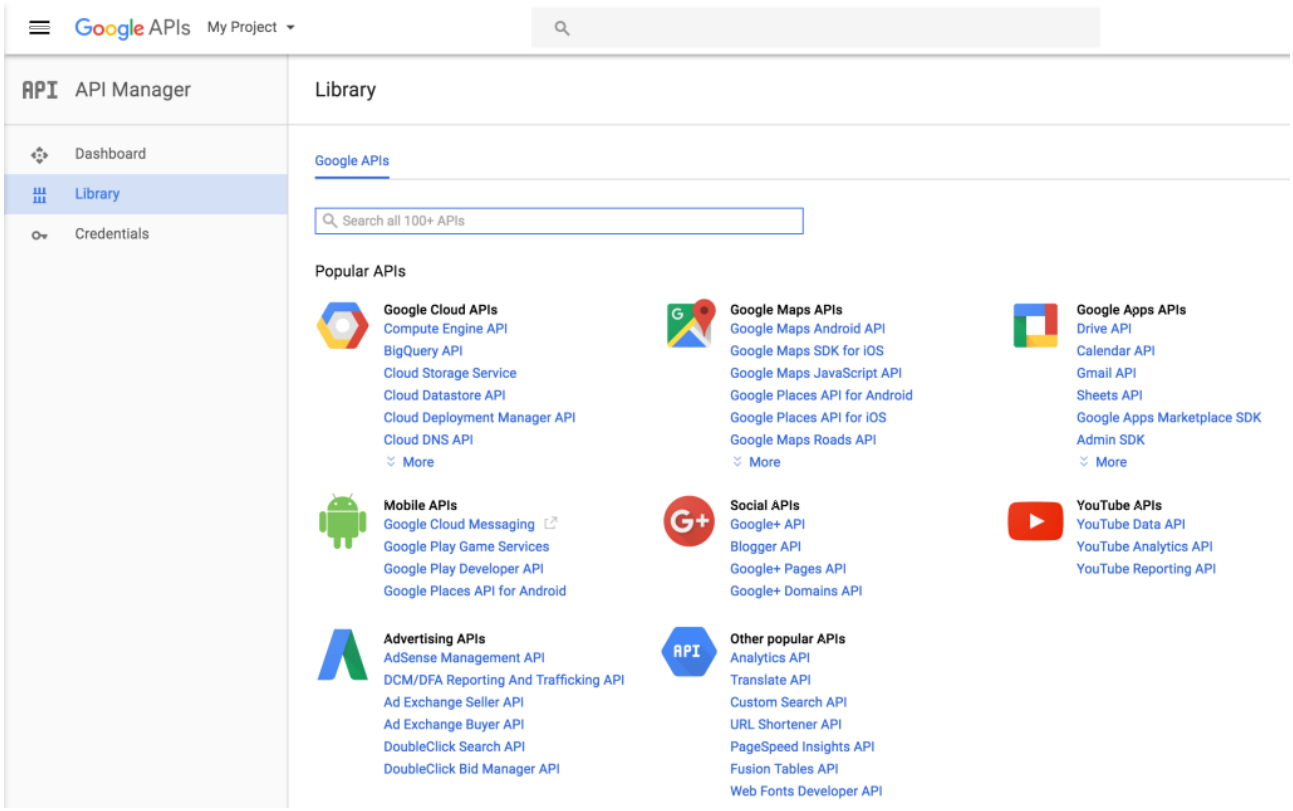
Finalment, la vostra clau apareixerà al taulell, com es mostra a la figura 1.14 a la secció de credencials.

FIGURA 1.14. Taulell de desenvolupadors que mostren la llista de claus



Cal destacar que seguint aquests passos s'activa automàticament l'API Google Maps per a JavaScript i es genera la clau apropiada. En cas de saltar-se algun pas o accedir directament al taulell s'haurà d'habilitar l'API manualment des de l'opció del menú ***Library*** (es pot veure el llistat a la figura 1.15) i generar les credencials des de l'opció del menú lateral ***Credentials***.

FIGURA 1.15. Taulell de desenvolupadors que mostren la biblioteca d'API



1.2.2 Creació d'un mapa simple

Un cop disposeu d'una clau, podeu començar a desenvolupar la vostra aplicació. Primerament haureu d'afegir la funció que serà l'encarregada de generar el mapa. Aquesta funció serà cridada automàticament en carregar-se la biblioteca de Google Maps.

Un exemple de funció d'inicialització, que generaria un nou mapa dins de l'element amb identificador `mapa`, centrat a la ciutat de Barcelona (que correspon a les coordenades especificades) i amb un zoom mitjà (el valor màxim per a la propietat `zoom` és 22), seria el següent:

```
new google.maps.Map(document.getElementById('mapa'), {
  center: {
    lat: 41.390205,
    lng: 2.154007
  },
  zoom: 11
});
```

Com es pot apreciar, el constructor rep dos arguments. El primer correspon al node on es vol dibuixar el mapa i el segon és un objecte amb les dades de configuració. En aquest cas s'ha especificat on s'ha de centrar el mapa (`center`) i quin nivell de zoom es desitja (`zoom`).

Podeu trobar més informació sobre l'element `script` i els seus atributs a l'enllaç següent: goo.gl/A7oxbg.

Seguidament s'ha de carregar la biblioteca de GoogleMaps, preferiblement utilitzant l'atribut `defer` ('diferir') per indicar que s'ha d'executar una vegada finalitzi

la càrrega del DOM. Juntament amb l'URL s'han d'incloure els paràmetres `callback` i `key`, que corresponen al nom de la funció a cridar per inicialitzar el mapa, i la vostra clau per a l'API Google Maps, generada a la consola de desenvolupadors de Google.

> La **biblioteca Google Maps** s'ha de carregar després que el vostre codi d'inicialització per evitar que sigui invocat per la biblioteca abans que s'hagi carregat.

Per exemple, si la funció que inicialitza els mapes s'anomena `iniciarAplicacioMapes` i la vostra clau és `AIzaSyDaYt85ZtjO2YDL-w94ZJLJE7F42HIr6Q0`, el codi per carregar la biblioteca seria el següent:

```
<script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDaYt85ZtjO2YDL-
    w94ZJLJE7F42HIr6Q0&callback=iniciarAplicacioMapes"></script>
```

Quant al codi HTML i CSS cal tenir en compte que caldrà utilitzar algun contenidor (per exemple, un element `div`) i especificar la mida que ha de tenir, tal com es pot veure en l'exemple següent, que mostra un mapa centrat a la ciutat de Barcelona que ocupa tota la pàgina:

```
<!doctype html>
<html>

<head>
  <meta charset="UTF-8">
  <title>Aplicació amb Google Maps</title>
  <style>
    html,body {
      height: 100%;
      margin: 0;
      padding: 0;
    }

    #mapa {
      height: 100%;
    }
  </style>
</head>

<body>

  <div id="mapa"></div>

  <script>
    iniciarMapa = function() {
    new google.maps.Map(document.getElementById('mapa'), {
      center: {
        lat: 41.390205,
        lng: 2.154007
      },
      zoom: 11
    });
  }
  </script>

  <script defer src="https://maps.googleapis.com/maps/api/js?key=
      AIzaSyDaYt85ZtjO2YDL-w94ZJLJE7F42HIr6Q0&callback=iniciarMapa"></script>

</body>
```

```
39
40 </html>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/apQvKa.

Fixeu-vos que aquest exemple és molt limitat: com que no s'ha guardat cap referència al mapa no és possible manipular-lo programàticament. Afortunadament, només cal referenciar el mapa generat amb una variable per poder accedir-hi, per exemple:

```
var mapa = new google.maps.Map(document.getElementById('mapa'), {
  center: {
    lat: 41.390205,
    lng: 2.154007
  },
  zoom: 11
});
```

1.2.3 Marcadors

Una de les accions més habituals en els mapes és afegir-hi marcadors. Per afegir-los-hi es disposa de dos sistemes:

- Especificar a quin mapa està associat el constructor i assignar-lo a la propietat `map`.
- Invocar el mètode `setMap` del marcador generat prèviament i passar el mapa com a argument.

Es pot trobar més informació sobre els marcadors a l'enllaç següent: goo.gl/dgAJVs.

A continuació podeu veure un exemple de creació d'un marcador en què s'ha especificat la posició del marcador, el mapa on s'ha de visualitzar i un títol que es mostra quan s'hi posa el cursor del ratolí:

```
var marcador = new google.maps.Marker({
    position: {
      lat: 41.375106,
      lng: 2.168342
    },
    map: mapa,
    title: "Seu central"
  });
```

Podeu veure aquest exemple a l'enllaç següent: codepen.io/ioc-daw-m06/pen/bgQEoY.

Una altra opció és crear primer els marcadors i assignar el mapa només en el moment en què es vulguin mostrar. Aquesta és l'opció utilitzada en el següent exemple, en el qual s'ha aplicat el disseny descendent per dividir l'aplicació en diferents funcions. Per una banda es crea el mapa i, per l'altra, un conjunt de

marcadors que són emmagatzemats en un *array*. Una vegada s'acaben de generar es recorre l'*array* i s'afegeixen al mapa:

```
<!doctype html>
<html>

<head>
  <meta charset="UTF-8">
  <title>Aplicació amb Google Maps</title>

  <style>
    html,
    body {
      height: 100%;
      margin: 0;
      padding: 0;
    }

    #mapa {
      height: 100%;
    }
  </style>
</head>

<body>

  <div id="mapa"></div>

  <script>
    var mapa;
    var marcadors = [];

    var iniciarMapa = function() {
      mapa = new google.maps.Map(document.getElementById('mapa'), {
        center: {
          lat: 41.390205,
          lng: 2.154007
        },
        zoom: 7
      });
    }

    var crearMarcadors = function() {
      marcadors.push(crearMarcador("IOC - Seu central", 41.375106, 2.168342));
      marcadors.push(crearMarcador("Proves Barcelona", 41.3860669, 2.1145104));
      marcadors.push(crearMarcador("Proves Girona", 41.961293, 2.829387));
      marcadors.push(crearMarcador("Proves Lleida", 41.623023, 0.6236748));
      marcadors.push(crearMarcador("Proves Tarragona", 41.120293, 1.247892));
    }

    var crearMarcador = function(titol, latitud, longitud) {
      var marcador = new google.maps.Marker({
        position: {
          lat: latitud,
          lng: longitud
        },
        title: titol
      })

      return marcador;
    }

    var afegirMarcadors = function() {
      for (var i = 0; i < marcadors.length; i++) {
        console.log("Afegir marcador: " + i);
        afegirMarcador(marcadors[i]);
      }
    }

    var afegirMarcador = function(marcador) {
```

```
      marcador.setMap(mapa);
    }

    var iniciarAplicacio = function() {
      iniciarMapa();
      crearMarcadors();
      afegirMarcadors();
    }
  </script>

  <script defer src="https://maps.googleapis.com/maps/api/js?key=
      AIzaSyDaYt85ZtjO2YDL-w94ZJLJE7F42HIr6Q0&callback=iniciarAplicacio"></
      script>
</body>

</html>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/KarMag.

Com es pot apreciar, una vegada es carrega la biblioteca de Google Maps, s'invoca la funció `iniciarAplicació`, que al seu torn crida les funcions `iniciarMapa` per afegir el mapa a la pàgina, `crearMarcadors` per generar els marcadors que s'emmagatzemmen a l'*array* `marcadors` i, finalment, la funció `afegirMarcadors`, que els afegeix a la pàgina.

S'ha embolcallat la creació de marcadors dins de la funció `crearMarcador`, que espera com a paràmetres un títol, la latitud i la longitud. D'aquesta manera es pot invocar la funció passant només aquests tres arguments per generar-lo (per exemple: `crearMarcador(''IOC - Seu central'', 41.375106, 2.168342)`.

Fixeu-vos que, partint d'aquesta implementació, és força senzill modificar-la per generar els marcadors a partir d'una estructura de dades, tal com es pot apreciar en l'exemple següent, en el qual s'ha modificat la funció `crearMarcadors` per utilitzar l'*array* `dades`, que conté la informació necessària.

```
var dades = [{
  seu: "IOC – Seu Central",
  lat: 41.375106,
  lon: 2.168342
}, {
  seu: "Proves Barcelona",
  lat: 41.3860669,
  lon: 2.1145104
}, {
  seu: "Proves Girona",
  lat: 41.961293,
  lon: 2.829387
}, {
  seu: "Proves Lleida",
  lat: 41.623023,
  lon: 0.6236748
}, {
  seu: "Proves Tarragona",
  lat: 41.120293,
  lon: 1.247892
}];

var crearMarcadors = function() {
  for (var i = 0; i < dades.length; i++) {
    marcadors.push(crearMarcador(dades[i].seu, dades[i].lat, dades[i].lon));
  }
}
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/egQzPV.

Cal tenir en compte que en aquest cas l'origen de les dades és un *array* d'objectes JavaScript que s'ha assignat directament al codi, però el funcionament de l'aplicació seria el mateix si les dades s'obtinguessin d'una altra font, per exemple, com a resposta d'una petició AJAX.

1.2.4 Finestres d'informació

Habitualment els marcadors mostren informació ampliada quan s'hi clica, però aquesta informació no es pot afegir directament als marcadors. Per una banda cal crear la finestra d'informació (`InfoWindow`) i, per l'altra, s'hi ha d'afegir un detector d'esdeveniments.

La creació de les finestres d'informació és força simple, només cal especificar-ne el contingut: una cadena de text en format HTML. És a dir, una finestra d'informació pot contenir enllaços, imatges i qualsevol tipus de contingut HTML.

En cas de treballar amb un únic marcador, n'hi hauria prou amb un codi com el següent:

```
1  var crearMarcador = function(dada) {
2    var finestraInfo = new google.maps.InfoWindow({
3      content: '<h3>' + dada.seu + '</h3><p><b>Adreça:</b>' + dada.adreca + '</p>
           '
4    });
5
6    var marcador = new google.maps.Marker({
7      position: {
8        lat: dada.lat,
9        lng: dada.lon
10     },
11     map: mapa,
12     title: dada.seu,
13   });
14
15   marcador.addListener('click', function() {
16     finestraInfo.open(mapa, this);
17   });
18 }
```

Les clausures es tracten a la unitat "Estructures definides pel programador" d'aquest mateix mòdul.

Cal destacar que, en afegir el detector d'esdeveniments, es crea una clausura. És a dir, quan es crida la funció associada a l'*event* clic, aquesta continua tenint accés al context concret en el qual s'ha generat i, per consegüent, té accés a l'objecte `finestraInfo` (que conté la informació de la finestra d'informació) que li correspon independentment de què s'hagin creat múltiples marcadors.

Vegeu a continuació com s'ha afegit l'adreça al conjunt de dades i com es guarda la referència de cada finestra per poder fer que es tanquin totes abans d'obrir-ne una de nova:

```
1  var mapa;
2  var dades = [{
```

```
  seu: "IOC – Seu Central",
  lat: 41.375106,
  lon: 2.168342,
  adreca: "Avinguda del Paral·lel, 71. Barcelona"
}, {
  seu: "Proves Barcelona",
  lat: 41.3860669,
  lon: 2.1145104,
  adreca: "C/ John M. Keynes, 1–11. Barcelona"
}, {
  seu: "Proves Girona",
  lat: 41.961293,
  lon: 2.829387,
  adreca: "Carrer de la Universitat de Girona, 10. Girona",
}, {
  seu: "Proves Lleida",
  lat: 41.623023,
  lon: 0.6236748,
  adreca: "Pi i Margall, 51. Lleida",
}, {
  seu: "Proves Tarragona",
  lat: 41.120293,
  lon: 1.247892,
  adreca: "Av. de Catalunya, 35. Tarragona",
}];
var marcadors = [];
var finestresInfo = [];

var iniciarMapa = function() {
  mapa = new google.maps.Map(document.getElementById('mapa'), {
    center: {
      lat: 41.390205,
      lng: 2.154007
    },
    zoom: 7
  });
}

var crearMarcadors = function() {
  for (var i = 0; i < dades.length; i++) {
    var marcador = crearMarcador(dades[i]);
    marcadors.push(marcador);
  }
}

var crearMarcador = function(dada) {
  var finestraInfo = new google.maps.InfoWindow({
    content: '<h3>' + dada.seu + '</h3><p><b>Adreça:</b>' + dada.adreca + '</p>
        '
  });

  var marcador = new google.maps.Marker({
    position: {
      lat: dada.lat,
      lng: dada.lon
    },
    title: dada.seu,
  });

  marcador.addListener('click', function() {
    tancarTotesLesFinestres();
    finestraInfo.open(mapa, this);
  });

  finestresInfo.push(finestraInfo);

  return marcador;
}

var tancarTotesLesFinestres = function() {
```

```
72    for (var i = 0; i < finestresInfo.length; i++) {
73      finestresInfo[i].close();
74    }
75  }
76
77  var afegirMarcadors = function() {
78    for (var i = 0; i < marcadors.length; i++) {
79      console.log("Afegir marcador: " + i);
80      afegirMarcador(marcadors[i]);
81    }
82  }
83
84  var afegirMarcador = function(marcador) {
85    marcador.setMap(mapa);
86  }
87
88  var iniciarAplicacio = function() {
89    iniciarMapa();
90    crearMarcadors();
91    afegirMarcadors();
92  }
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/apQmgO.

S'ha afegit l'*array* `finestresInfo` per guardar la referència de totes les finestres creades i la funció `tancarTotesLesFinestres`. Aquesta funció es crida en fer clic sobre un marcador i s'encarrega de recórrer aquest *array* per invocar el mètode `close` de cadascuna de les finestres d'informació.

Fixeu-vos que a la funció `crearMarcador` s'ha implementat la creació de la finestra i s'ha afegit la invocació a la funció `tancarTotesLesFinestres` a la funció invocada en detectar-se l'*event* `click`. A més a més, cada finestra creada es guarda a l'*array* `finestresInfo`.

Si en lloc d'afegir la creació de les finestres d'informació a la funció `crearMarcador` ho feu a la funció `crearMarcadors`, us trobareu amb un error important, ja que en crear-se la clausura entre la funció invocada en fer clic i la funció `crearMarcador` sempre s'accedirà a l'última finestra generada. Això es pot apreciar en l'exemple següent, en el qual s'han modificat només les funcions `afegirMarcador` i `afegirMarcadors`:

```
1   var crearMarcadors = function() {
2     for (var i = 0; i < dades.length; i++) {
3       var marcador = crearMarcador(dades[i]);
4       marcadors.push(marcador);
5
6       var finestraInfo = new google.maps.InfoWindow({
7         content: '<h3>' + dades[i].seu + '</h3><p><b>Adreça:</b>' + dades[i].
              adreca + '</p>'
8       });
9
10      marcador.addListener('click', function() {
11        window.tancarTotesLesFinestres();
12        finestraInfo.open(mapa, this);
13      });
14
15      finestresInfo.push(finestraInfo);
16    }
17  }
18
19  var crearMarcador = function(dada) {
```

```
  var marcador = new google.maps.Marker({
    position: {
      lat: dada.lat,
      lng: dada.lon
    },
    title: dada.seu,
  });

  return marcador;
}
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/MJzEwM.

Com es pot apreciar, totes les finestres d'informació mostren les dades de l'últim element creat. Com que el context de totes les funcions de detecció és el mateix (la funció `crearMarcadors` només ha estat invocada una vegada) totes accedeixen a la mateixa variable `finestraInfo`, que contindrà l'últim valor assignat; és a dir, la finestra amb la informació `Proves Tarragona`.

Fixeu-vos que, en cas d'afegir la finestra d'informació a la funció `crearMarcador`, s'invoca una vegada per a cada marcador i, conseqüentment, el context d'execució de cada funció de detecció és únic.

S'ha d'anar amb compte quan es treballa amb clausures i detecció d'*events*: s'hi poden cometre fàcilment i són difícils de depurar.

1.2.5 Cas pràctic: integrar una font de dades obertes amb Google Maps

En aquest exemple es treballaran els punts següents:

- Ús de la biblioteca jQuery.
- Ús de la biblioteca Google Maps.
- Cerca de serveis Open Data.
- Realització de consultes AJAX.
- Càrrega d'un mapa de Google Maps.
- Creació de marcadors.
- Creació de finestres d'informació.

Quan es necessita treballar amb mapes, el més habitual és que la informació que s'hagi de mostrar s'obtingui d'una font de dades externa, com per exemple un servei web propi o dades obertes ofertes per tercers mitjançant AJAX.

Cal recordar que en el cas de dades de tercers, només seran accessibles si el servidor implementa mecanismes CORS o JSONP, ja que en cas contrari els navegadors bloquejaran la petició AJAX.

Una d'aquestes fonts de dades obertes es troba a la pàgina de l'Observatori de dades culturals de Barcelona (barcelonadadescultura.bcn.cat/dades-obertes), on podeu trobar la documentació de l'API (dades.eicub.net/doc).

A partir de la informació que es troba en aquests enllaços, és possible realitzar consultes a l'API per obtenir diferents llistats d'informació; per exemple, per obtenir informació sobre els cinemes només cal cercar a la documentació en format XML (dades.eicub.net/doc) la paraula *cinemes* i trobareu les dades següents:

```
<dataset>
  <name>cinemes_dadesglobals</name>
  <url>http://dades.eicub.net/api/1/cinemes_dadesglobals</url>
  <title>Dades globals dels cinemes</title>
  <description>Dades globals dels cinemes de la ciutat</description>
  <columns>
    <column>Any</column>
    <column>Dada</column>
    <column>Valor</column>
    <column>Nota</column>
  </columns>
</dataset>
<dataset>
  <name>cinemes_inventari</name>
  <url>http://dades.eicub.net/api/1/cinemes_inventari</url>
  <title>Inventari Cinemes</title>
  <description>Inventari de les sales de cinema</description>
  <columns>
    <column>Districte</column>
    <column>Equipament</column>
    <column>Titularitat</column>
    <column>Latitud</column>
    <column>Longitud</column>
    <column>Web</column>
  </columns>
</dataset>
```

És a dir, hi ha dos conjunts de dades referents a cinemes: un que mostra les dades globals de cada cinema de la ciutat i un altre que mostra el llistat de sales de cinema. Com es pot apreciar, per accedir a aquestes dades cal utilitzar l'URL que s'indica, que per al llistat de cinemes és http://dades.eicub.net/api/1/cinemes-inventari.

Fixeu-vos que a la documentació s'indica una sèrie de columnes per a cada conjunt de dades. Aquestes dades es corresponen amb la informació proporcionada pel llistat corresponent per a cadascun dels cinemes. Així doncs, a l'"Inventari Cinemes" es poden trobar les coordenades per localitzar els cinemes sobre el mapa.

A partir d'aquesta informació es podria decidir implementar una aplicació que mostrés a un mapa els punts on es troben tots els cinemes de Barcelona, utilitzant la informació proporcionada de la manera següent:

- **Equipament**: com a títol del marcador i capçalera de la finestra d'informació.
- **Districte**, **Titularitat** i **Web**: llistada a la finestra d'informació.
- **Latitud** i **Longitud**: utilitzades per crear el marcador.

Així doncs, una possible implementació d'aquest mapa seria la que es troba a continuació. Les dades es carreguen mitjançant AJAX des de l'URL dades.eicub.net/api/1/cinemes-inventari i una vegada descarregades es creen els marcadors i finestres d'informació:

```
<!doctype html>
<html>

<head>
  <meta charset="UTF-8">
  <title>Localització de cinemes a la ciutat de Barcelona</title>

  <style>
    html, body {
    height: 100%;
    margin: 0;
    padding: 0;
  }

  #mapa {
    height: 100%;
  }

  ul {
    list-style-type: none;
    padding: 0;
  }

  li b {
    display: inline-block;
    width: 75px;
  }
</style>
</head>

<body>

  <div id="mapa"></div>
  <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>

  <script>
    var mapa;
    var dades;
    var marcadors = [];
    var finestresInfo = [];

    var iniciarMapa = function() {
      mapa = new google.maps.Map(document.getElementById('mapa'), {
        center: {
          lat: 41.390205,
          lng: 2.154007
        },
        zoom: 13
      });
    }

    var crearMarcadors = function() {
      for (var i = 0; i < dades.length; i++) {
        var marcador = crearMarcador(dades[i]);
        marcadors.push(marcador);
      }
    }

    var tancarTotesLesFinestres = function() {
      for (var i = 0; i < finestresInfo.length; i++) {
        finestresInfo[i].close();
      }
    }

```

```
    var crearMarcador = function(dada) {
      var finestraInfo = new google.maps.InfoWindow({
        content: '<h3>' + dada.Equipament + '</h3>' +
          '<ul>' +
          '<li><b>Districte:</b>' + dada.Districte + '</li>' +
          '<li><b>Titularitat:</b>' + dada.Titularitat + '</li>' +
          '<li><b>Web:</b><a href="' + dada.Web + '">' + dada.Web + '</a></li>'
               +
          '</ul>'
      });

      var marcador = new google.maps.Marker({
        position: {
          lat: parseFloat(dada.Latitud),
          lng: parseFloat(dada.Longitud)
        },
        title: dada.Equipament,
      });

      marcador.addListener('click', function() {
        tancarTotesLesFinestres();
        finestraInfo.open(mapa, this);
      });

      finestresInfo.push(finestraInfo);

      return marcador;
    }

    var afegirMarcadors = function() {
      for (var i = 0; i < marcadors.length; i++) {
        afegirMarcador(marcadors[i]);
      }
    }

    var afegirMarcador = function(marcador) {
      marcador.setMap(mapa);
    }

    var iniciarMarcadors = function() {
      crearMarcadors();
      afegirMarcadors();
    }

    var iniciarAplicacio = function() {
      iniciarMapa();
      carregarDades();
    }

    var carregarDades = function() {
      $.ajax('http://dades.eicub.net/api/1/cinemes_inventari', {
        dataType: 'json',
        success: function(resposta) {
          dades = resposta;
          iniciarMarcadors();
        }
      });
    }
  </script>
  <script defer src="https://maps.googleapis.com/maps/api/js?key=
      AIzaSyDaYt85ZtjO2YDL_w94ZJLJE7F42HIr6Q0&callback=iniciarAplicacio"></
      script>
</body>

</html>
```

Podeu veure aquest exemple a l'enllaç següent: codepen.io/ioc-daw-m06/pen/rjQpjQ.

Fixeu-vos que aquesta aplicació té unes fortes dependències i si es canviés l'ordre d'execució, l'aplicació no funcionaria. Per una banda, la biblioteca jQuery s'ha de carregar abans que s'iniciï l'aplicació, perquè la càrrega de dades en depèn, ja que s'ha utilitzat per simplificar la crida AJAX. Per altra banda, la funció `iniciarAplicació` ha d'estar disponible abans que acabi de carregar la biblioteca de Google Maps, ja que per inicialitzar els mapes cridarà aquesta funció.

Les dades han de ser carregades abans de poder inicialitzar els marcadors i després que s'hagi carregat la biblioteca de Google Maps. Per aquesta raó, s'invoca `carregarDades` dintre de la funció `iniciarAplicació`, que és invocada per la biblioteca Google Maps i, una vegada es rep la resposta a la crida AJAX, els marcadors s'inicialitzen invocant la funció `iniciarMarcadors`.

Quant a l'aspecte de l'aplicació, s'ha modificat el codi CSS per estendre el mapa a tota la pantalla i donar un format més atractiu a les finestres d'informació. Tingueu en compte que les finestres accepten tot tipus de codi HTML i, per tant, és possible afegir altres elements com, per exemple, imatges.

Les possibilitats que ofereix l'API de Google Maps és molt extensa i es recomana consultar els tutorials i la documentació oficial que podeu trobar a goo.gl/B8veWw en cas de necessitar funcionalitats més complexes, com per exemple afegir altres controls al mapa o personalitzar la icona del marcador.

1.3 Desenvolupament de jocs amb HTML5

A finals de l'any 2016 la majoria de fabricants de navegadors (Google i Mozilla entre altres) van deixar d'admetre la utilització de connectors per a Flash o per a les miniaplicacions (*applets*) de Java als navegadors. Per una banda, no es podia garantir la seguretat d'aquests connectors i, per l'altra, amb les característiques afegides a HTML5 ja no eren necessaris.

'Applet' o miniaplicació

Un *applet*, en català 'miniaplicació', és un petit programa desenvolupat en Java perquè s'executi al navegador.

S'ha de tenir en compte que quan es parla d'**HTML5** no només es tracta del llenguatge de marques HTML, sinó que engloba també CSS, JavaScript i les noves Web API afegides al llenguatge (com per exemple els elements `àudio`, `vídeo`, `canvas`, `API WebSockets`...).

Quant al desenvolupament de jocs amb HTML5, l'element clau va ser la inclusió de l'element `canvas` i, en menor mesura, els elements `àudio` i `vídeo`. El primer permet dibuixar formes i imatges dintre de l'element, mentre que els altres dos permeten reproduir àudio i vídeo directament al navegador.

Tot i que hi ha motors molt populars per desenvolupar jocs amb HTML5, com Construct 2 (www.scirra.com) o ImpactJs (impactjs.com), és relativament senzill desenvolupar un joc només utilitzant JavaScript, HTML i CSS.

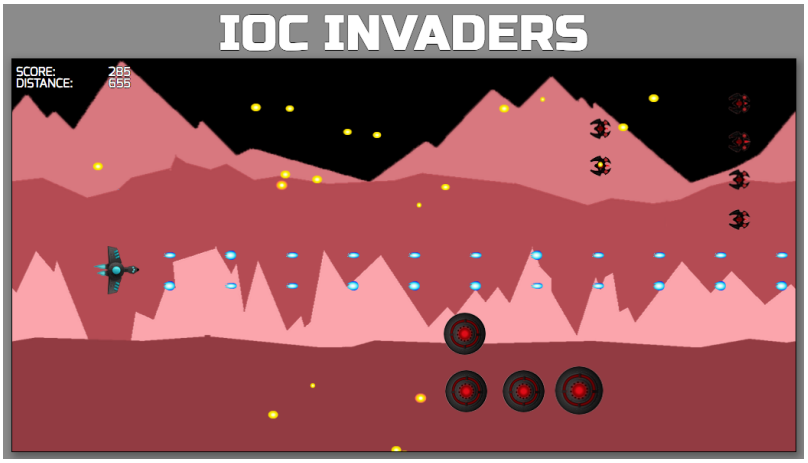
Cal destacar que molts d'aquests motors prometen exportar els jocs a diferents plataformes (Android, iOS, tvOS, etc.); en realitat, el que fan és generar una aplicació que conté una finestra de navegador sense cap botó, on executen el joc en JavaScript.

Per altra banda, hi ha altres motors de joc multiplataforma que es programen en altres llenguatges però exporten a JavaScript. Per exemple, el motor Unity (unity3d.com/es) permet desenvolupar jocs amb C# o JavaScript i exportar-los a HTML5 mitjançant un reproductor que utilitza l'API WebGL per renderitzar gràfics en 2D i 3D.

1.3.1 Introducció

Un exemple relativament senzill de com desenvolupar un joc amb JavaScript és l'*IOC Invaders* (vegeu la figura 1.16). Aquest joc permet il·lustrar com es pot estructurar el codi i aprofitar les característiques afegides a HTML5 (concretament els elements `canvas` i `àudio`) per crear un joc de naus amb múltiples nivells, tipus d'enemics i armes.

Figura 1.16. Joc "IOC Invaders" en funcionament



Podeu trobar el codi complet d'aquest joc a l'enllaç següent: github.com/XavierGaro/ioc-invaders. Tots els recursos són lliures, tant el codi com els recursos gràfics i d'àudio, però si els utilitzeu en els vostres projectes, heu d'incloure els enllaços a les fonts i mencionar-ne els autors.

Cal destacar la utilització del `canvas` tant per dibuixar la nau del jugador, els enemics, les bales i les explosions com per simular un desplaçament horitzontal (*scroll*, en anglès) amb quatre nivells de profunditat. Tot i que es fa servir contínuament la mateixa imatge, aquest sistema es podria millorar per utilitzar un *array* d'imatges per concatenar.

Un punt a tenir molt en compte quan es desenvolupen jocs és que s'han d'optimitzar per obtenir-ne el millor rendiment possible. Per exemple, en altres tipus d'aplicacions gestionar la creació i destrucció d'objectes no és crític perquè

l'execució del recol·lector de brossa és inapreciable. En canvi, quan s'han de gestionar centenars de naus, bales, efectes de fons i control del jugador, l'execució del recol·lector provoca una baixada de rendiment crítica i molt evident.

A *IOC Invaders*, per optimitzar l'ús de la memòria i reduir les crides al recol·lector de brossa, s'ha utilitzat una tècnica coneguda com a `pool`, que consisteix a reutilitzar els mateixos elements en lloc de destruir-los i crear-ne de nous. Això permet mantenir un mínim de 60 FPS (*frames* o fotogrames per segon), és a dir, l'element `canvas` es redibuixa com a mínim 60 vegades per segon.

Quant a la gestió de nivells i recursos s'han utilitzat fitxers de configuració en format JSON que es carreguen mitjançant AJAX. D'aquesta manera, sense modificar el codi del joc es poden modificar els nivells i els enemics, afegir-ne de nous o actualitzar-ne els recursos (imatges i sons).

Regles del joc

Abans de començar a programar cal tenir clares quines seran les regles del joc. Quan s'acaba la partida? Com s'arriba al final del nivell? Què passa si s'arriba al final del joc?

En el cas de l'*IOC Invaders* s'ha decidit utilitzar el següent conjunt de regles:

- Tant les naus de l'enemic com la del jugador són destruïdes si xoquen amb l'adversari.
- Tant les bales de l'enemic com les del jugador són destruïdes si impacten en un adversari.
- Qualsevol nau impactada per una bala de l'adversari és destruïda.
- Quan la nau del jugador és destruïda el joc s'acaba.
- Quan la nau d'un enemic és destruïda augmenta la puntuació del jugador.
- Els enemics apareixen quan s'ha recorregut una distància determinada del mapa que s'estableix per a cada nivell.
- El nivell es considera per finalitzat quan s'arriba a una distància determinada i no és visible cap enemic.
- Quan es finalitza l'últim nivell es torna al primer (conservant la puntuació).

Presentació del joc

Tot i que l'acció de qualsevol joc es trobarà representada dins del `canvas`, cal recordar que l'entorn d'execució és el navegador i, per consegüent, es té accés tant a HTML com a CSS. Això permet, per exemple, mostrar el títol del joc a la capçalera, els crèdits i les instruccions sota del joc i fins i tot utilitzar les puntuacions com etiquetes HTML sobre el `canvas`.

Per descomptat, es poden utilitzar efectes i animacions de CSS, per exemple, per mostrar un efecte de *fade out* (fos a negre) o desplaçament de lletres o nombres (puntuació actual, vides restants, missatges de felicitació, etc.).

A *IOC Invaders* s'ha aprofitat el fitxer HTML per afegir la capçalera del joc, els crèdits corresponents als recursos gràfics i d'àudio emprats i la llicència corresponent al peu, com es pot veure en el codi corresponent al fitxer `ioc-invaders.html`:

```
<!DOCTYPE html>
<html lang="ca">
<head>
    <meta charset="UTF-8">
    <title>IOC Invaders</title>
    <link href="https://fonts.googleapis.com/css?family=Russo+One" rel="
        stylesheet">
    <link href="https://fonts.googleapis.com/css?family=Exo" rel="stylesheet">
    <link href="css/style.css" rel="stylesheet">
</head>
<body>

<h1>IOC INVADERS</h1>

<div id="game-background">
</div>

<div class="credits">
    <h2>CRÈDITS</h2>
    <div>
        <h3>Música</h3>
        <ul>
            <li>Defense Line: <a target="_blank" href="https://www.dl-sounds.
                com/royalty-free/defense-line/">Background
                Loop</a></li>
            <li>Star Commander1(<a target="_blank" href="https://www.dl-sounds.
                com/royalty-free/star-commander1/"> DL
                Sounds</a>)
            </li>
        </ul>
    </div>

    <div>
        <h3>Efectes de so</h3>
        <ul>
            <li>Laser: <a target="_blank" href="http://soundbible.com/1087-
                Laser.html"> SoundBible.com</a></li>
            <li>Explosions: inferno (<a target="_blank" href="http://www.
                freesound.org/people/inferno/sounds/18384/">
                freesound.org</a>)
            </li>
            <li>Altres – Xavier Garcia Rogríguez</li>
        </ul>
    </div>

    <div>
        <h3>Gràfics</h3>
        <ul>
            <li>Naus: sujit1717 (<a target="_blank"
                                    href="http://opengameart.org/content/
                                        complete-spaceship-game-art-pack">
                                        Space Odyssey
                Pack</a>)
            </li>
            <li>Fons i trets: Xavier Garcia Rogríguez</li>
        </ul>
    </div>
</div>

<div class="footer">
```

```
    Xavier Garcia Rodríguez 2017. <a href="http://www.apache.org/licenses/
        LICENSE-2.0" target="_blank">Llicencia Apache
    2.0</a>. Podeu trobar el codi font d'aquest joc a <a target="_blank"
                                                        href="https://github.
                                                            com/XavierGaro/
                                                            ioc-invaders">
                                                            GitHub</a>.
</div>

<script src="js/ioc-invaders.js">
</script>

</body>
</html>
```

També s'han aprofitat les característiques de CSS3 per afegir alguns efectes de transició que són activats pel joc, i s'han utilitzat dues fonts externes:

- Vora negra al voltant de tot el text.
- Ombra al voltant del `canvas`.
- Missatges que es fan visibles i s'esvaeixen, així com pantalles de transició (classe `fadeable`).
- Utilització de les fonts Russo One i Exo facilitades per Google Fonts.
- Utilització de diferents colors per a diferents elements.

A continuació podeu trobar el codi corresponent al fitxer "css/style.css" que inclou tots els estils emprats:

```
body {
    color: white;
    background-color: gray;
    text-align: center;
    font-family: 'Russo One', sans-serif;
    text-shadow: 1px 0 0 #000, 0 -1px 0 #000, 0 1px 0 #000, -1px 0 0 #000;
}

canvas {
    margin: 0 auto;
    background: transparent;
    position: absolute;
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;
}

h1, h2, h3 {
    margin: 0;
    line-height: 1;
}

h1 {
    font-size: 64px;
}

h2 {
    color: red;
    padding: 15px;
    font-size: 50px;
}
```

```
33
34  h3 {
35      font-size: 40px;
36      color: #FFD700;
37  }
38
39  ul {
40      list-style: none;
41      padding: 0;
42      text-align: center;
43      color: white;
44  }
45
46  li {
47      font-family: 'Exo', sans-serif;
48  }
49
50  a {
51      color: cornflowerblue;
52      text-decoration: none;
53      border-bottom: 1px dotted;
54  }
55
56  #game-background {
57      position: relative;
58      margin: 0 auto;
59      width: 1024px;
60      height: 512px;
61      border: 1px solid black;
62      -webkit-box-shadow: 3px 5px 10px 2px rgba(0, 0, 0, 0.39);
63      -moz-box-shadow: 3px 5px 10px 2px rgba(0, 0, 0, 0.39);
64      box-shadow: 3px 5px 10px 2px rgba(0, 0, 0, 0.39);
65      background-color: black;
66  }
67
68  #background {
69      z-index: -2;
70  }
71
72  .ui {
73      margin: 0 auto;
74      width: 1024px;
75      position: relative;
76      top: 0;
77  }
78
79  .score, .distance {
80      position: absolute;
81      top: 5px;
82      left: 5px;
83      color: #FFF;
84      cursor: default;
85      width: 150px;
86      text-align: left;
87      font-family: 'Exo', sans-serif;
88  }
89
90  .score span, .distance span {
91      float: right;
92  }
93
94  .distance {
95      top: 20px;
96  }
97
98  .game-over, .messages {
99      position: absolute;
100     top: 180px;
101     left: 0;
102     right: 0;
```

```
    color: red;
    font-size: 40px;
    cursor: default;
}

.game-over span, .messages span {
    font-size: 20px;
    position: relative;
}

.game-over span {
    cursor: pointer;
}

.game-over {
    z-index: 100;
    display: none;
    cursor: default;
}

.game-over span:hover, .messages {
    color: #FFD700;
}

.fadeable {
    -webkit-transition: opacity 3s ease-in-out;
    -moz-transition: opacity 3s ease-in-out;
    -ms-transition: opacity 3s ease-in-out;
    -o-transition: opacity 3s ease-in-out;
    opacity: 0;
}

.credits {
    margin: 0 auto;
    width: 1024px;
}

.credits div {
    width: 33%;
    float: left;
}

.footer {
    font-size: smaller;
    font-family: 'Exo', sans-serif;
    position: fixed;
    bottom: 0;
    width: 100%;
    text-align: center;
}
```

El resultat d'aplicar aquesta estructura HTML i els estils la podeu veure a la figura 1.17.

FIGURA 1.17. Representació de l'estructura HTML i CSS del joc "IOC Invaders"

Per facilitar la portabilitat del joc és recomanable que tots els elements imprescindibles per al seu funcionament es creïn mitjançant JavaScript. Així només cal comprovar que l'identificador del contenidor en el qual s'afegirà el joc és correcte. Per exemple, en el joc *IOC Invaders* només cal afegir un element amb l'identificador `game-background` i la resta d'elements es generaran automàticament.

Concretament, els elements necessaris per al funcionament del *IOC Invaders* s'afegeixen en finalitzar la càrrega de la pàgina mitjançant el codi següent, corresponent al final del fitxer `ioc-invader.js`:

```
1  window.onload = function () {
2      var gameContainer = document.getElementById('game-background'),
3          canvas,
4          game;
5
6      gameContainer.innerHTML = '' +
7          '<canvas id="game-canvas" width="1024" height="512" class="fadeable">'
               +
8          'Your browser does not support canvas. Please try again with a
               different browser.' +
9          '</canvas>' +
10         '<div class="ui">' +
11         '   <div class="score">SCORE: <span id="score"></span></div>' +
12         '   <div class="distance">DISTANCE: <span id="distance"></span></div>'
               +
13         '</div>' +
14         '<div class="game-over" id="game-over">GAME OVER<p><span>Restart</span
               ></p></div>' +
15         '<div class="messages fadeable" id="messages"></div>';
16
17     canvas = document.getElementById('game-canvas');
18
19     IOC_INVADERS.start({
20         "asset_data_url": "asset-data.json",
21         "entity_data_url": "entity-data.json",
22         "levels_data_url": "level-data.json"
```

```
    }, canvas);

    document.getElementById('game-over').addEventListener('click', IOC_INVADERS
        .restart);
};
```

Com es pot apreciar, s'obté la referència a l'element `game-background` i s'estableix com a contingut intern d'aquest element el codi HTML que conté el `canvas` i els elements que mostraran les puntuacions i els missatges.

A més a més s'obté la referència al `canvas` amb el qual s'inicialitza el joc, juntament amb la ruta de les dades corresponents als recursos i les entitats i els nivells que formaran part del joc. Per acabar, s'afegeix un detector de l'*event* `click` a l'element `game-over` per reinicialitzar el joc.

1.3.2 Encapsulament del joc

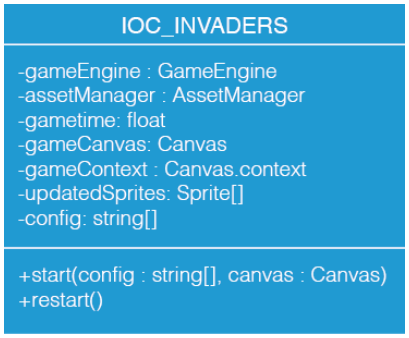
Per encapsular la informació és recomanable fer servir el patró de disseny *mòdul*. D'aquesta manera el joc no interfereix amb altres aplicacions i totes les seves funcions, mètodes i objectes queden aïllats de l'usuari, que no podrà manipular-lo, ja que només tindrà accés a les funcions públiques exposades pel mòdul.

En el cas del joc *IOC Invaders*, el mòdul disposa d'una sèrie de propietats internes, funcions constructores per a pràcticament tots els components del joc i dues funcions públiques que són accessibles externament i permeten iniciar el joc i reiniciar la partida: `start` i `restart`. A continuació, podeu trobar el codi del mòdul sense incloure els constructors de classes internes, i a la figura **??** podeu veure el diagrama UML corresponent:

```
var IOC_INVADERS = function () {
    var GameState = {
            GAME_OVER: "GameOver",
            LOADING_NEXT_LEVEL: "LoadingNextLevel",
            RUNNING: "Running"
        },

        MAX_TIME_OUTSIDE_BOUNDARIES = 180,
        gameCanvas,
        config,
        gameEngine,
        assetManager,
        gametime,
        gameContext,
        updatedSprites = [];

  // Resta de constructors i funciones internes

  return {
    start: function (conf, canvas) {
      config = conf;
      gameCanvas = canvas;
      gameManager = gameEngineConstructor();
      assetManager = assetManagerConstructor(function (current, total) {
        console.log("Carregats:" + current + "/" + total);
      });

      gameEngine.init();
```

```
29      },
30
31      restart: function () {
32        gameEngine.restart();
33      }
34    }
35  }();
```

Figura 1.18. Diagrama UML corresponent al mòdul IOC_INVADERS



Cal destacar que un dels objectes declarats és `GameState` i té un format que pot resultar-vos peculiar. Tot i que a JavaScript les enumeracions no existeixen (`enums` en altres llenguatges), és possible emular-les amb estructures com aquesta.

Podeu trobar més informació sobre els *enums* a l'enllaç següent: goo.gl/qnhKSC.

En programació no es recomana mai fer servir *nombres màgics* al codi (valors literals sense explicacions o en múltiples llocs): per una banda, és difícil saber a què fan referència i, per l'altra, és possible que estiguin duplicats en diversos punts del codi i, llavors, si cal canviar-los, resulta complicat.

Per resoldre aquest problema s'acostumen a fer servir constants o enumeracions. Quan els valors no estan relacionats entre ells és recomanable utilitzar constants (a *IOC Invaders* s'utilitza la constant `MAX_TIME_OUTSIDE_BOUNDARIES`, per exemple) però quan hi ha una relació entre ells és recomanable utilitzar una enumeració (`GameState` fa referència als estats possibles del joc).

A JavaScript les constants no existeixen, però es poden simular posant els noms en majúscules com a convenció.

Utilitzant constants i enumeracions és molt fàcil fer el canvi dels valors, ja que només s'ha de fer en un punt. A més a més, la majoria d'entorns de desenvolupament habiliten l'autocompletar i el ressaltat d'errors, si no s'ha escrit correctament el nom, cosa que no passa quan es treballa amb nombres i cadenes de text. A més a més, el codi és més entenedor; per exemple: no aporta la mateixa informació `var a = 100;` (a què fa referència 100?, és molt?, és poc?, és correcte?) que `var a = MAX_SPEED;`, que genera molts menys dubtes.

Tornant al tema de l'encapsulament, en algunes ocasions ens pot interessar fer tot el contrari: com per exemple estendre objectes propis de JavaScript per afegir funcionalitats que no es troben en el llenguatge. A continuació podem veure com s'amplia el prototip dels `arrays` afegint el mètode `contains` perquè comprovi si un element es troba dins de l'*array*:

```
1  /**
2   * @param {*} needle – objecte a cercar dins de l'array
3   * @returns {boolean} – cert si es troba o false en cas contrari
4   */
```

```
Array.prototype.contains = function (needle) {
    for (var i in this) {
        if (this[i] == needle) return true;
    }
    return false;
};
```

Aquesta funció s'utilitza dintre de l'*IOC Invaders* per comprovar si un `sprite` ja ha estat actualitzat (es troba dins de l'*array* `updatedSprites`) o no.

Un altre exemple d'extensió seria afegir la funció `clamp` (disponible en altres llenguatges) que permet *encaixonar* un valor entre un mínim i un màxim, de manera que el resultat estigui sempre dintre d'aquests límits:

```
/*
 * @param {number} min – valor mínim
 * @param {number} max – valor màxim
 * @returns {number} – El nombre si està dins del limit o el valor corresponent
      al límit
 */
Number.prototype.clamp = function (min, max) {
    return Math.min(Math.max(this, min), max);
};
```

Un exemple d'utilització d'aquesta funció es troba en l'objecte `player`, que es fa servir per evitar que la nau del jugador surti de la pantalla:

```
that.position.x = that.position.x.clamp(0, gameCanvas.width – that.sprite.size.
    width);
that.position.y = that.position.y.clamp(0, gameCanvas.height – that.sprite.size
    .height);
```

Ara compareu-lo amb el codi sense utilitzar el mètode `clamp`; s'entén fàcilment però és molt llarga:

```
if (that.position.x <0) {
  that.position.x = 0;
} else if (that.position.x > gameCanvas.width – that.sprite.size.width) {
  that.position.x = gameCanvas.width – that.sprite.size.width;
}

if (that.position.y < 0) {
  that.position.y = 0;
} else if (that.position.y > gameCanvas.height – that.sprite.size.height) {
  that.position.y = gameCanvas.height – that.sprite.size.height;
}
```

I, finalment, compareu-lo amb la implementació fent servir els mètodes `Math.max` i `Math.min`, no és tan curta com amb el mètode `clamp` i és més complicada d'entendre:

```
that.position.x = Math.min(Math.max(that.position.x, 0), gameCanvas.width –
    that.sprite.size.width);
that.position.y = Math.min(Math.max(that.position.y, 0), gameCanvas.height –
    that.sprite.size.height);
```

S'ha de tenir en compte que habitualment no es recomana modificar el prototipus dels objectes del llenguatge, però hi ha ocasions en què és molt útil i permet escriure un codi més concís i més entenedor.

1.3.3 Gestió de les dades

Habitualment, quan es desenvolupa un joc s'ha de treballar amb dades que han de ser manipulades i reutilitzades. Per exemple, la informació sobre els tipus d'enemics o els tipus d'armes que es poden trobar, la informació sobre les peces d'un puzle... Aquesta informació es pot tractar com un catàleg només de consulta: a partir d'aquesta informació es generen els elements del joc i no s'ha de modificar.

Aquestes dades poden provenir de fonts externes (per exemple, carregades mitjançant AJAX) o poden estar incrustades al codi. Es pot considerar que aquestes dades són un dipòsit o un catàleg, a partir del qual es construeixen els objectes que s'utilitzaran en el joc.

En molts casos un dipòsit de dades no es limita a emmagatzemar dades, sinó que n'acostuma a realitzar algun tipus de tractament abans d'afegir-les o recuperar-les. El seu funcionament és molt similar al del patró de disseny factoria, però amb una implementació més simple.

Podeu trobar més informació sobre el patró de disseny factoria a l'enllaç següent: goo.gl/r3sBkz.

Per exemple, a *IOC Invaders* s'han implementat dos dipòsits de dades (*respository*, en anglès) diferents. Per una banda, s'ha creat un dipòsit d'entitats que s'encarrega de gestionar les dades de les entitats que formen el joc (naus, bales, explosions...) i, per l'altra, un dipòsit d'estratègies que permet assignar diferents funcions de moviment a les entitats (concretament a les bales i les naus enemigues).

A continuació podeu veure com s'ha implementat el dipòsit d'*entitats* (es tracta d'un nom escollit arbitràriament que fa referència a les dades que s'utilitzen per crear instàncies d'objectes al joc, vegeu la figura 1.19), que pot rebre un *array* d'entitats (per exemple, carregades des d'un fitxer d'entitats) i en cas necessari assigna la funció `move` obtinguda del dipòsit d'estratègies:

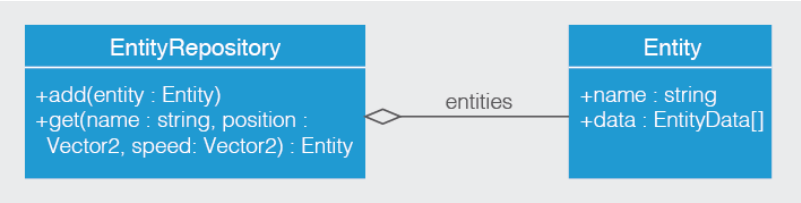
Crear instàncies

En el context de JavaScript, *crear instàncies* o en alguns llocs 'instanciar' (de l'anglès *instance*), consisteix a crear un objecte nou (una instància) a partir d'un altre objecte. És a dir, el nou objecte tindrà els mateixos mètodes i propietats públics de l'objecte original.

```
1  var entitiesRepository = (function () {
2    var entities = {};
3
4    function addEntity(name, data) {
5      var entity = data;
6
7      if (data.move) {
8        entity.move = strategiesRepository.get(data.move);
9      }
10
11     entities[name] = entity;
12   }
13
14   return {
15     add: function (entity) {
16       if (Array.isArray(entity)) {
17         for (var i = 0; i < entity.length; i++) {
18           addEntity(entity[i].name, entity[i].data);
19         }
20       } else {
21         addEntity(entity.name, entity.data);
22       }
23     },
24
25     get: function (name, position, speed) {
26       var entity = entities[name];
```

```
        if (!entity) {
          console.error("No es troba la entitat: ", entity);
        }
        entity.position = position;
        entity.speed = speed;

        return entity;
    }
  }
})();
```

Figura 1.19. Diagrama UML dipòsit d'entitats



Fixeu-vos que es tracta d'una funció autoinvocada i, per conseqüent, només exposa els mètodes de l'objecte que retorna (`add` i `get`). Així s'encapsulen les dades dintre del dipòsit i s'assegura que només s'accedirà als elements de la forma prevista.

Al contrari que en el dipòsit anterior, el dipòsit d'estratègies no obté les dades d'una font externa, sinó que conté un diccionari de funcions que correspon a les diverses estratègies que poden assignar-se a les entitats, com es pot veure a continuació:

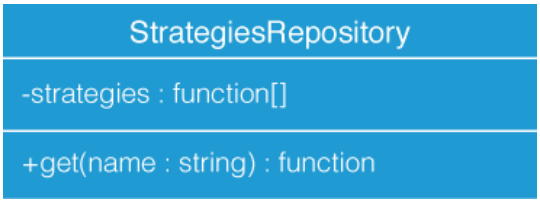
```
var strategiesRepository = (function () {
  var strategies = {

    movement_pattern_a: function () {
      if (!this.extra.ready) {
        this.extra.speed = Math.max(Math.abs(this.speed.x), Math.abs(this.speed
            .y));
        this.extra.leftEdge = this.position.x - 10 * this.extra.speed;
        this.extra.rightEdge = this.position.x + 10 * this.extra.speed;
        this.extra.topEdge = this.position.y + 10 * this.extra.speed;
        this.extra.bottomEdge = this.position.y - 10 * this.extra.speed;
        this.extra.direction = {x: this.speed.x <= 0 ? -1 : 1, y: 1};
        this.extra.ready = true;
      }

      this.position.x += this.speed.x;
      this.position.y += this.speed.y * this.extra.direction.y;


      if (this.position.y > this.extra.topEdge || this.position.y < this.extra.
          bottomEdge) {
        this.speed.x = this.extra.direction.x >= 0 ? this.extra.speed : -this.
            extra.speed;
        this.speed.y = 0;
        this.extra.direction.y = -this.extra.direction.y;
      }

      if (this.position.x <= this.extra.leftEdge) {
        this.speed.x = 0;
        this.speed.y = this.extra.speed;
        this.extra.leftEdge = this.position.x - 10 * this.extra.speed;
      } else if (this.position.x >= this.extra.rightEdge) {
        this.speed.x = 0;
```

```
31          this.speed.y = this.extra.speed;
32          this.extra.rightEdge = this.position.x + 10 * this.extra.speed;
33        }
34
35        this.position.y = this.position.y.clamp(this.extra.bottomEdge, this.extra
              .topEdge);
36      },
37
38      movement_pattern_b: function () {
39        this.position.x += this.speed.x;
40        this.position.y += this.speed.y;
41      },
42
43      movement_pattern_c: function () {
44        if (!this.extra.ready) {
45          this.extra.age = 0;
46          this.extra.speed = Math.max(Math.abs(this.speed.x), Math.abs(this.speed
                .y));
47          this.extra.ready = true;
48          this.extra.vertical = this.speed.x > this.speed.y;
49        }
50
51        if (this.extra.direction === 1) {
52          this.speed.x = this.extra.speed * Math.cos(-this.extra.age * Math.PI /
                64);
53        } else {
54          this.speed.y = this.extra.speed * Math.sin(this.extra.age * Math.PI /
                64);
55        }
56
57        this.extra.age++;
58        this.position.x += this.speed.x;
59        this.position.y += this.speed.y;
60      },
61
62      movement_pattern_d: function () {
63        if (!this.extra.ready) {
64          this.extra.age = 0;
65          this.extra.speed = Math.max(Math.abs(this.speed.x), Math.abs(this.speed
                .y));
66          this.extra.ready = true;
67          this.extra.vertical = this.speed.x > this.speed.y;
68        }
69
70        if (this.extra.direction === 1) {
71          this.speed.x = this.extra.speed * Math.cos(this.extra.age * Math.PI /
                64);
72        } else {
73          this.speed.y = this.extra.speed * Math.cos(this.extra.age * Math.PI /
                64);
74        }
75
76        this.extra.age++;
77        this.position.x += this.speed.x;
78        this.position.y += this.speed.y;
79      }
80    };
81
82    return {
83      get: function (strategy) {
84        return strategies[strategy];
85      }
86    }
87  }());
```

Tot i que el codi pot intimidar una mica, fixeu-vos que només consisteix en un diccionari de dades al qual corresponen funcions que determinen una nova posició. Com és d'esperar, es tracta d'una funció autoinvocada i per tant només exposa el

mètode `get`, que permet obtenir la funció corresponent a l'estratègia de moviment amb aquest nom (vegeu la figura 1.20).

FIGURA 1.20. Diagrama UML dipòsit d'estratègies



Fixeu-vos que tots dos dipòsits disposen d'un diccionari de dades privat (objecte de JavaScript que fa servir una cadena de text com a clau) per emmagatzemar les dades i un mètode `get` públic per recuperar-les.

1.3.4 Interacció amb l'aplicació

Els dos dispositius més habituals per interactuar amb una aplicació són el ratolí i el teclat. Per estrany que pugui semblar, tots dos poden ser problemàtics a l'hora de treballar amb ells donades les discrepàncies que es poden trobar entre els navegadors.

En el cas del ratolí, la dificultat més gran que es presenta en treballar amb l'element `canvas` és determinar sobre quin punt es troba realment el cursor, ja que no hi ha cap mètode 100% fiable per accedir a aquesta informació i cada navegador implementa diferents solucions.

El més recomanable és treballar amb les propietats `offsetX` i `offsetY`, ja que tot i que són experimentals formen part de les recomanacions del W3C. Aquestes propietats retornen la posició del cursor respecte a l'element clicat, que correspondrà al `canvas`.

Quant a l'ús del teclat, hi ha problemes similars. Cada navegador ha realitzat la implementació d'una manera diferent i no es garanteix que totes les tecles responguin de la mateixa manera als *events* `keydown`, `keyup` i `keypress`. Tampoc no es retornen les mateixes propietats amb l'*event*, de manera que alguns navegadors retornen el codi a la propietat `keyCode` mentre que d'altres utilitzen `charCode`.

Podeu trobar més informació sobre les propietats dels *events* disparats pel ratolí a l'enllaç següent: goo.gl/t0ijCe.

Per unificar aquestes entrades, la solució és crear un objecte que centralitzi l'entrada dels dispositius i que l'aplicació consulti a aquest objecte en lloc de gestionar directament els *events*. Aquest objecte també pot encarregar-se de gestionar les diferències entre navegadors, de manera que tots els canvis que calgui fer sobre aquest aspecte es poden fer des d'un mateix punt.

Per exemple, a *IOC Invaders* això s'aconsegueix a través de l'objecte `inputController`, una funció autoinvocada que encapsula la complexitat de detectar quines tecles s'han premut, i exposa només dues propietats que permeten consultar l'estat de les tecles fent servir cadenes de text: `space`, `left`, `up`, `right` i `down`:

```
var inputController = (function () {
  var KEY_CODES = {
    32: 'space',
    37: 'left',
    38: 'up',
    39: 'right',
    40: 'down'
  },

  KEY_STATUS = {};

  for (var code in KEY_CODES) {
    KEY_STATUS[KEY_CODES[code]] = false;
  }

  document.onkeydown = function (e) {
    var keyCode = (e.keyCode) ? e.keyCode : e.charCode;

    if (KEY_CODES[keyCode]) {
      e.preventDefault();
      KEY_STATUS[KEY_CODES[keyCode]] = true;
    }
  };

  document.onkeyup = function (e) {
    var keyCode = (e.keyCode) ? e.keyCode : e.charCode;

    if (KEY_CODES[keyCode]) {
      e.preventDefault();
      KEY_STATUS[KEY_CODES[keyCode]] = false;
    }
  };
  return {
    KEY_CODES: KEY_CODES,
    KEY_STATUS: KEY_STATUS
  }
})();
```

Com es pot apreciar, s'utilitzen dos diccionaris:

- **KEY_CODES**: relaciona el codi corresponent a cada tecla amb un nom més entenedor pel desenvolupador.
- **KEY_STATUS**: emmagatzema l'estat de cada tecla (`true` si s'ha premut o `false` en cas contrari).

Aquests diccionaris són exposats com a propietats públiques de l'objecte, mentre que la implementació que gestiona els canvis queda amagada gràcies a la clausura creada per la funció autoinvocada. D'aquesta manera és possible consultar l'estat de les tecles, però no poden modificar-se externament.

Exposar una propietat o mètode

En programació orientada a objectes, *exposar* fa referència a mètodes o propietats a les quals es pot accedir des d'altres punts del programari, i la resta de la implementació queda inaccessible.

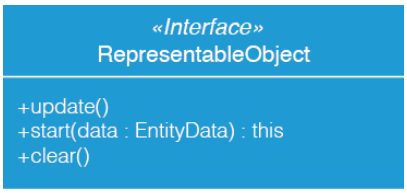
Per consultar l'estat concret d'un botó, només cal consultar el valor del diccionari `KEY_STATUS`. Per exemple, per saber l'estat de la tecla corresponent a la fletxa esquerra (anomenada `left` en el codi), només cal consultar l'estat del diccionari de la manera següent: `inputController.KEY_STATUS.left`. Aquesta consulta retornarà `true` o `false` segons si s'ha premut el botó o no.

1.3.5 Elements representables al joc

Tots els jocs tenen elements que han de ser representats a la pantalla: fons del joc, jugadors, peces d'un puzle, enemics, bales, efectes... Tots aquests elements tenen una característica en comú: han de poder mostrar-se a la pantalla i han de poder-se eliminar.

Per simplificar la implementació pot utilitzar-se una interfície (vegeu la figura 1.21), que serà implementada per cadascun d'aquests elements. S'ha de disposar d'un mètode d'actualització (*update* en anglès), que serà cridat cada vegada que es redibuixa el `canvas`, un mètode per inicialitzar l'element (*start* en anglès) i un mètode per netejar (*clear* en anglès) quan calgui eliminar-lo.

FIGURA 1.21. Diagrama UML interfície RepresentableObject



Cal destacar que JavaScript no admet l'ús d'interfícies i, per tant, `RepresentableObject` no es troba definit al codi del joc, però forma part del disseny de l'aplicació.

Podeu trobar més informació sobre la utilització de l'element `canvas` a l'enllaç següent: goo.gl/GV9awc.

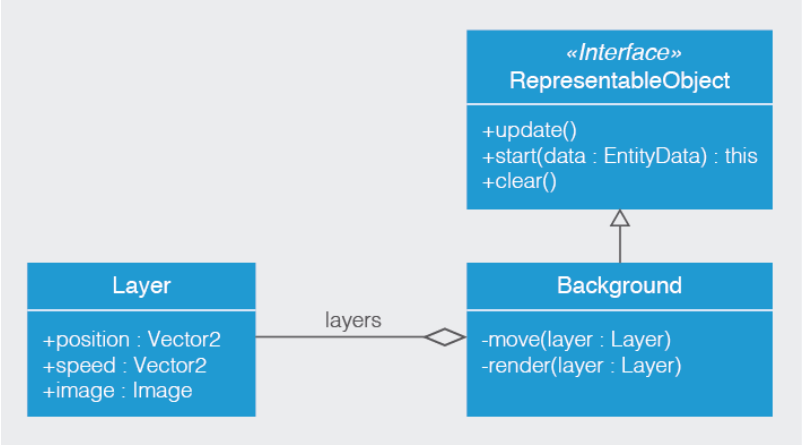
L'element `canvas` permet dibuixar tant gràfics en 2D com en 3D mitjançant l'API WebGL, tot i que aquest últim sistema no es fa servir gaire perquè és molt més complex. Tot i així, hi ha motors de jocs com Unity que permeten exportar jocs en 3D i sí que aprofiten aquesta API.

El context 2D del `canvas` permet escriure textos i dibuixar formes o imatges completes carregades a la memòria (fitxers d'imatge, per exemple). S'ha de tenir en compte que el contingut és estàtic, és a dir, els continguts no són animats, es crea aquesta il·lusió reescrivint-los en diferents posicions o canviant la imatge dibuixada.

`Vector2` és una estructura de dades que representa un vector de dues dimensions amb les propietats: `x` i `y`.

Per exemple, per dibuixar el fons del joc *IOC Invaders*, s'utilitza l'objecte generat per `backgroundConstructor`, que emmagatzema la informació sobre 4 imatges que es dibuixen en ordre com si fossin capes (vegeu la figura 1.22). La primera capa que es dibuixa queda al fons de la pila, la segona capa es dibuixa a sobre, i així contínuament.

Figura 1.22. Diagrama UML objecte de tipus Background i les seves associacions



A continuació podeu veure el codi del generador d'objectes de tipus `Background` utilitzat a *IOC Invaders*, que inclou un fons amb desplaçament infinit amb 4 nivells de profunditat:

```
var backgroundConstructor = function () {
  var that = {},
    layers = {};

  function move(layer) {
    layer.position.x += layer.speed.x;
    layer.position.y += layer.speed.y;

    if (layer.position.x < -layer.image.width || layer.position.x > layer.image
        .width) {
      layer.position.x = 0;
    }

    if (layer.position.y < -layer.image.height || layer.position.y > layer.
        image.height) {
      layer.position.y = 0;
    }

  }

  function render(layer) {
    // Imatge actual
    gameContext.drawImage(
      layer.image, layer.position.x, layer.position.y, layer.image.width, layer
          .image.height);

      // Imatge següent
      gameContext.drawImage(
        layer.image, layer.position.x + layer.image.width - 1,
        layer.position.y, layer.image.width, layer.image.height);
    }

    that.update = function () {
      for (var i = 0; i < layers.length; i++) {
        move(layers[i]);
        render(layers[i]);
      }
    };

    that.start = function (data) {
      layers = data.layers;

      for (var i = 0; i < layers.length; i++) {
        layers[i].image = assetManager.getImage(layers[i].id);
        layers[i].position = {x: 0, y: 0}
```

```
43        }
44      };
45
46      that.clear = function () {
47        layers = {};
48        speed.reset();
49      };
50
51    return that;
52  };
```

Com podeu veure, per construir l'objecte s'ha fet servir el patró funcional. És a dir, la funció `backgroundConstructor` retorna un objecte "Background". Aquest objecte exposa uns mètodes públics (afegits a l'objecte `that`, que és retornat) i encapsula les propietats (declarades mitjançant la paraula `var` al principi de la funció) i mètodes privats (declarats amb la paraula clau `function`) per distingir-los més clarament.

Cada vegada que s'invoca el mètode `update`, primer s'invoca el mètode privat `move`, que desplaça la posició de cada capa segons la velocitat establerta i, seguidament, s'invoca el mètode `render`, que les redibuixa en la nova posició mitjançant el mètode del context `drawImage`, que requereix unes coordenades, la mida i la imatge a dibuixar.

Fixeu-vos que es dibuixa la mateixa imatge dos cops, una a continuació de l'altra. D'aquesta manera s'aconsegueix un efecte de bucle, ja que no es pot apreciar on acaba una i on comença la següent. Aquest efecte es podria millorar incloent imatges més llargues o fent servir un *array* d'imatges que anés alternant-se.

En aquest cas, com que la imatge de fons ocupa tot el `canvas` no cal netejar-lo, però hi ha casos en què és convenient utilitzar el mètode `clearRect` del context per eliminar el contingut anterior (totalment o parcialment).

Cal destacar que les imatges utilitzades s'obtenen d'un gestor de recursos (l'objecte `assetManager`) mitjançant el mètode `getImage`. D'aquesta manera, l'objecte `Background` no necessita saber com ni on s'ha generat, només ha de passar al mètode l'identificador de la imatge que necessita.

Sprites

Un altre element clau per representar a la pantalla són els *sprites*. S'acostuma a utilitzar el mateix terme, sigui una sola imatge o un conjunt que forma una animació. En el cas de constar de múltiples imatges, poden dibuixar-se seqüencialment per simular diferents moviments. Per exemple, la nau que es mostra a la figura 1.23 simula l'efecte dels propulsors fent augmentar i disminuir la flama.

Figura 1.23. Sprite d'un alien de tipus A del joc "IOC Invaders"

Un ***sprite*** és un mapa de bits bidimensional utilitzats per primera vegada a les consoles i ordenadors de 8 bits.

Es pot trobar més informació sobre els *sprites* a l'enllaç següent: goo.gl/55RIoP.

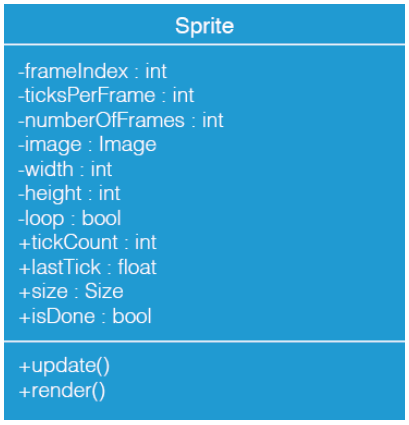
En jocs més avançats, cada personatge o objecte pot tenir associats múltiples *sprites*, de manera que pot haver-hi diferents animacions per a les accions que es poden portar a terme: atacar, caminar, córrer, saltar... I, per descomptat, tots els elements poden utilitzar-los: el personatge del jugador, enemics, bales, explosions, decoracions... Només cal fer una ullada als jocs d'ordinadors i videoconsoles de 8 i 16 bits (fins i tot de màquines recreatives) per veure'n les possibilitats.

Si un joc corre a 60 FPS, es mostrarà una imatge cada 16 mil·lisegons aproximadament.

La gestió d'aquests elements és més complicada que en el cas dels fons perquè requereixen càlculs addicionals per determinar si s'han de reproduir infinitament i quant de temps s'ha de mostrar cada imatge que compon l'animació.

A la figura 1.24 podeu veure el diagrama corresponent a un objecte `Sprite` generat per la funció generadora `spriteConstructor` del joc *IOC Invaders*, corresponent al codi següent:

Figura 1.24. Diagrama UML objecte de tipus Sprite




```
1  var spriteConstructor = function (options) {
2
3    var that = {},
4      frameIndex = 0,
5      ticksPerFrame = options.ticksPerFrame || 0,
6      numberOfFrames = options.numberOfFrames || 1,
7      image = options.image,
8      width = image.width,
9      height = image.height,
10     loop = options.loop === undefined ? true : options.loop;
11
12     that.tickCount = 0;
13     that.lastTick = gametime;
14     that.position = options.position || {x: 0, y: 0};
15     that.size = {width: width / numberOfFrames, height: height};
16     that.isDone = false;
17
18     that.update = function () {
19
20       if (updatedSprites.contains(that)) {
21         return;
22       } else {
23         updatedSprites.push(that);
24       }
```

```
25
26      that.tickCount++;
27      that.lastTick = gametime;
28
29      if (that.tickCount > ticksPerFrame) {
30        that.tickCount = 0;
31
32        if (frameIndex < numberOfFrames - 1) {
33          frameIndex += 1;
34        } else {
35          that.isDone = !loop;
36          frameIndex = 0;
37        }
38      }
39    };
40
41    that.render = function () {
42
43      if (that.isDone) {
44        return;
45      }
46
47      gameContext.drawImage(
48        image,
49        frameIndex * width / numberOfFrames,
50        0,
51        width / numberOfFrames,
52        height,
53        that.position.x,
54        that.position.y,
55        width / numberOfFrames,
56        height);
57    };
58
59  return that;
60 };
```

Com es pot apreciar, la tasca principal d'aquesta classe és determinar si s'ha completat l'animació i si cal repetir-la i dibuixar la imatge que correspongui. Fixeu-vos que en el cas que una imatge contingui una animació, aquesta no es dibuixa completa sinó que només se'n dibuixa el fragment corresponent.

En aquest cas, per simplificar, s'ha considerat que totes les imatges que formen l'animació tenen la mateixa mida; així doncs, només cal saber quin és el `frame` intern que cal dibuixar i calcular-ne la posició dins de la imatge.

Fixeu-vos que en el mètode `update` es comprova primer si aquest *sprite* ja ha estat actualitzat (aquesta propietat correspon al mòdul del joc) i, en cas contrari, s'afegeix a l'*array* abans de continuar amb l'actualització. Aquesta és una optimització que s'ha aplicat a aquest joc per evitar que s'actualitzin múltiples vegades per *frame* alguns elements.

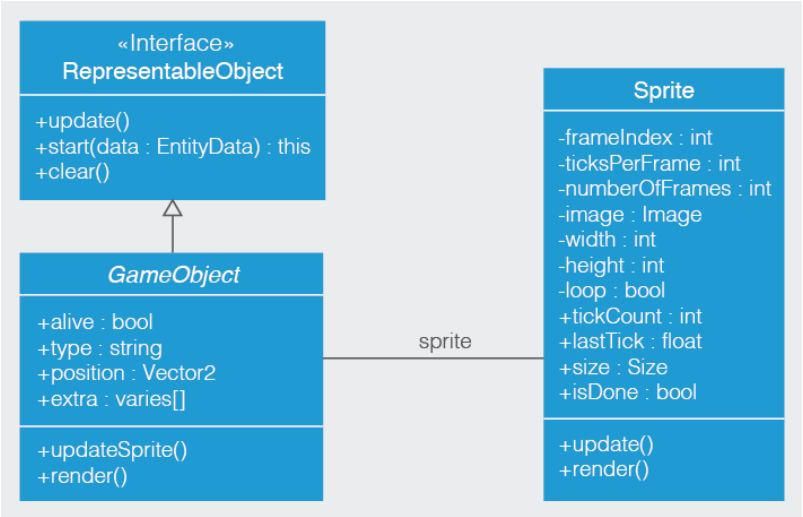
Objectes de joc: 'GameObject'

Alguns motors de desenvolupament de jocs, com Unity, utilitzen una classe especial d'objectes anomenats *objectes de joc* (*game objects* en anglès). Aquests objectes són elements que poden ser representats a la pantalla i que han de mantenir-se actualitzats contínuament. Això s'aconsegueix invocant el seu mètode `update` (entre d'altres) a cada iteració del bucle principal del joc.

Per exemple, un objecte de tipus bala canviarà la seva posició, actualitzarà l'animació de l'*sprite* que el representa, comprovarà si ha impactat a un adversari i si ha sortit de la pantalla; un enemic, a més a més, determinarà si cal disparar de nou, i una explosió només actualitzarà l'animació fins que aquesta acabi i desaparegui.

Tot i que els objectes de tipus `Background` també es poden considerar objectes de joc (corresponents als objectes `GameObject` que podeu veure a la figura 1.25), a *IOC Invaders* s'ha decidit diferenciar-los per generalitzar propietats comunes a: explosions, bales i naus.

Figura 1.25. Diagrama UML objecte de tipus GameObject i les seves relacions



Vegeu a continuació la implementació del generador d'objectes de tipus `Constructor` del joc *IOC Invaders*:

```
1  var gameObjectConstructor = function (options) {
2    var that = {
3      alive: false,
4      type: null,
5      position: null,
6      sprite: null,
7      extra: {}
8    };
9
10   that.updateSprite = function () {
11     that.sprite.position = that.position;
12     that.sprite.update();
13   };
14
15   that.render = function () {
16     that.sprite.render()
17   };
18
19   that.start = function (data) {
20     console.error("Error. Aquest mètode no està implementat");
21     return that;
22   };
23
24   that.update = function () {
25     console.error("Error. Aquest mètode no està implementat");
26   };
27
28   that.clear = function () {
```

```
    console.error("Error. Aquest mètode no està implementat");
  };

  return that;
};
```

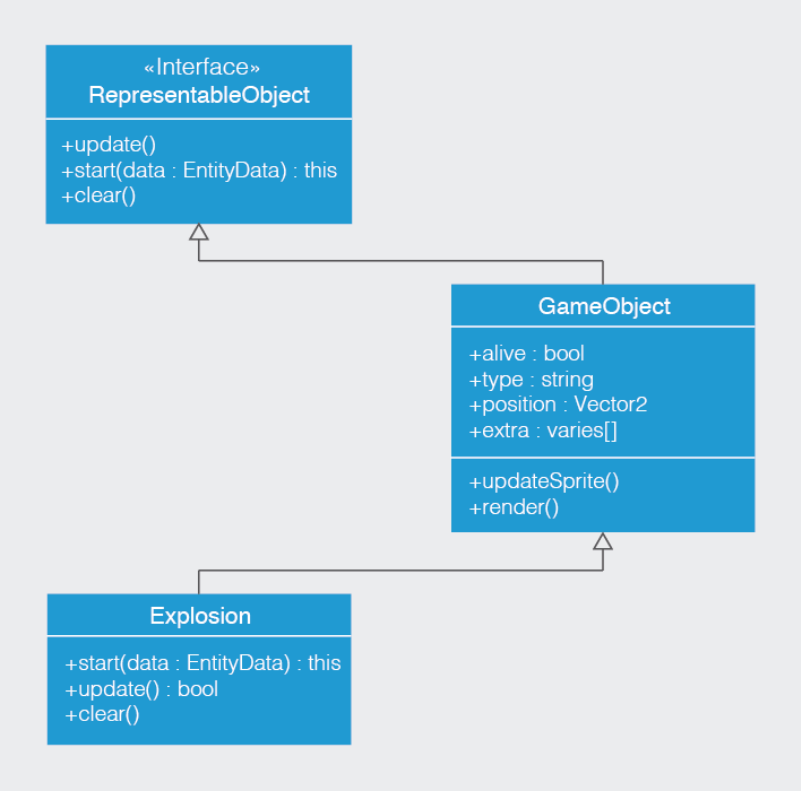
Cal destacar que `GameObject` s'ha de considerar una classe abstracta (tot i que a JavaScript no hi ha aquesta distinció). És a dir, no s'ha d'instanciar directament sinó que altres objectes l'utilitzen com a classe pare o superclasse.

Tipus especialitzats de 'GameObject' a "IOC Invaders"

Tot i que això no sempre és necessari, a *IOC Invaders* s'han generat múltiples generadors d'objectes per aprofitar la reutilització del codi. D'aquesta manera l'estructura és molt més clara i s'evita la duplicació de codi.

Per una banda, es van diferenciar les explosions d'altres elements del joc, ja que les característiques de les explosions són idèntiques a les de `GameObject` i es limita a implementar els mètodes `update`, `start` i `clear` de la interfície `RepresentableObject`, tal com es mostra a la figura 1.26. Podeu veure la implementació en el següent bloc codi:

Figura 1.26. Diagrama UML objecte de tipus Explosion i les seves relacions

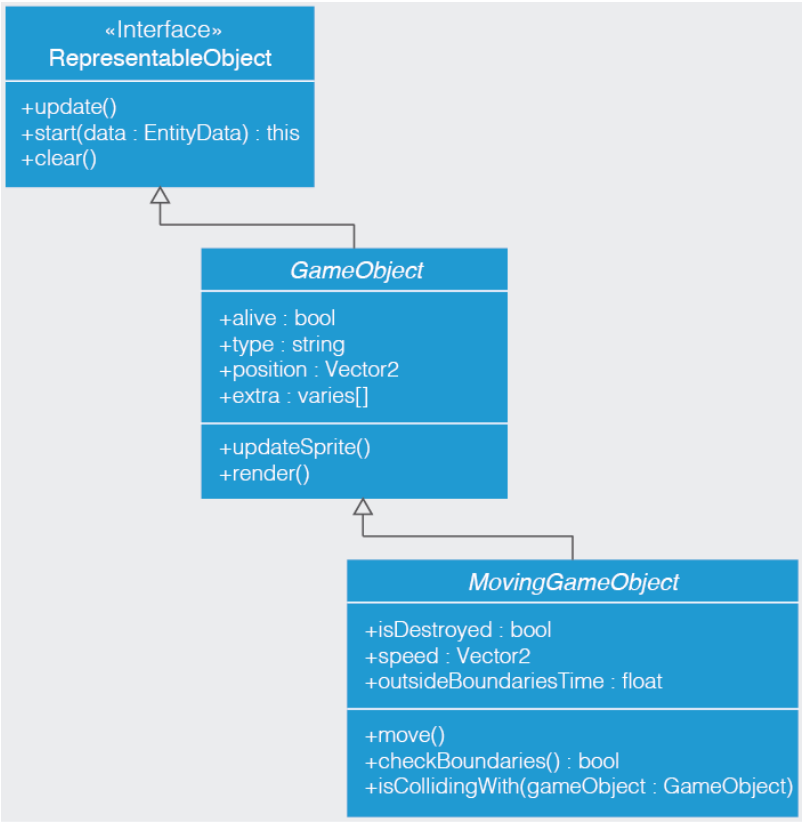



```
var explosionConstructor = function (options) {
  var that = gameObjectConstructor(options);

  that.start = function (data) {
    that.alive = true;
    that.type = data.type;
    that.position = data.position;
```

```
    that.sprite = assetManager.getSprite(data.sprite);
    that.sprite.isDone = false;
    assetManager.getSound(data.sound);
    return that;
  };

  that.clear = function () {
    that.alive = false;
    that.type = null;
    that.position = {x: 0, y: 0};
    that.sprite = null;
  };

  that.update = function () {
    if (that.sprite.isDone) {
      return true;
    }

    that.updateSprite();
    that.render();
  };

  return that;
};
```

Per altra banda, es troben els objectes que s'han de moure i, per conseqüent, han de controlar-se si es troben fora de la pantalla o si han col·lidit. Per generalitzar les característiques comunes d'aquest tipus d'objectes a *IOC Invaders* s'ha afegit un generador d'objectes de tipus `MovingGameObject` (veieu la figura 1.27) amb el codi que podeu trobar a continuació:

Figura 1.27. Diagrama UML objecte de tipus MovingGameObject i les seves relacions




```
var movingGameObjectConstructor = function (options) {
  var that = gameObjectConstructor(options);
```

```
3
4   that.isDestroyed = false;
5   that.speed = null;
6   that.outsideBoundariesTime = 0;
7
8   that.move = function () {
9     console.error("Error. Aquest mètode no està implementat");
10  };
11
12  that.checkBoundaries = function () {
13    if (this.position.x >= gameCanvas.width
14      || this.position.x <= -this.sprite.size.width
15      || this.position.y > gameCanvas.height
16      || this.position.y < -this.sprite.size.height) {
17        this.outsideBoundariesTime++;
18    } else {
19      this.outsideBoundariesTime = 0;
20    }
21
22    return that.outsideBoundariesTime >= MAX_TIME_OUTSIDE_BOUNDARIES;
23  };
24
25  that.isCollidingWith = function (gameObject) {
26    return (this.position.x < gameObject.position.x + gameObject.sprite.size.
          width
27      && this.position.x + this.sprite.size.width > gameObject.position.x
28      && this.position.y < gameObject.position.y + gameObject.sprite.size.
            height
29      && this.position.y + this.sprite.size.height > gameObject.position.y);
30  };
31
32  return that;
33 };
```

Fixeu-vos que els objectes generats s'han de tractar com si fossin abstractes, ja que requereix la implementació de la funció `move` a més a més dels mètodes `update`, `start` i `clear` de la interfície `RepresentableGameObject`.

Com es pot apreciar, els objectes que siguin herència del tipus `MovingGameObject` inclouen, a més del mètode per moure's, els mètodes `checkBoundaries` i `iscollisingWith` per comprovar si l'objecte del joc es troba dins dels límits de la pantalla o si ha col·lidit amb un altre objecte.

La resta de tipus d'objectes segueixen el mateix sistema, només cal destacar que tant els enemics com els jugadors són herència del mateix tipus d'objectes (el tipus `SpaceShip`), amb la diferència que les naus enemigues disparen aleatòriament i la nau del jugador es controla amb el teclat. A la figura 1.28 podeu veure la jerarquia completa d'objectes representables, i a continuació, el codi generador dels objectes de tipus `Shot`, `SpaceShip` i `Player`:

```
1  var shotConstructor = function (options) {
2    var that = movingGameObjectConstructor(options);
3
4    that.start = function (data) {
5      that.alive = true;
6      that.type = data.type;
7      that.position = data.position;
8      that.sprite = assetManager.getSprite(data.sprite);
9      assetManager.getSound(data.sound);
10     that.speed = data.speed;
11
12     // Dades i funcions específiques de cada tipus d'enemic
13     that.extra = data.extra || {};
```

```
    that.move = data.move.bind(that);
    that.outsideBoundariesTime = 0;

    return that;
  };

  that.clear = function () {
    that.isDestroyed = false;
    that.alive = false;
    that.outsideBoundariesTime = 0;
    that.type = null;
    that.position = {x: 0, y: 0};
    that.sprite = null;
    that.speed = {x: 0, y: 0};

    // Dades i funcions específiques de cada tipus d'enemic
    that.extra = {};
    that.move = null;
  };

  that.update = function () {
    if (that.isDestroyed) {
      return true;
    }

    that.updateSprite();
    that.move();

    if (that.checkBoundaries()) {
      return true;
    }

    that.render();
  };

  return that;
};

var spaceshipConstructor = function (options) {
  var that = movingGameObjectConstructor(options);

  that.bulletPool = options.pool.bullet;
  that.explosionPool = options.pool.explosion;

  that.fire = function () { // @protected
    for (var i = 0; i < that.cannon.length; i++) {
      that.shoot(that.cannon[i]);
    }
  };

  that.shoot = function (cannon) { // @protected
    var origin;

    if (Math.random() < cannon.fireRate / 100) {
      origin = {x: that.position.x + cannon.position.x, y: that.position.y +
          cannon.position.y};
      that.bulletPool.instantiate(cannon.bullet, origin, cannon.direction);
    }
  };

  that.start = function (data) {
    that.alive = true;
    that.type = data.type;
    that.position = data.position;
    that.sprite = assetManager.getSprite(data.sprite);
    that.cannon = data.cannon;
    that.explosion = data.explosion;
    that.speed = data.speed;
    that.points = data.points;
    that.outsideBoundariesTime = 0;
```

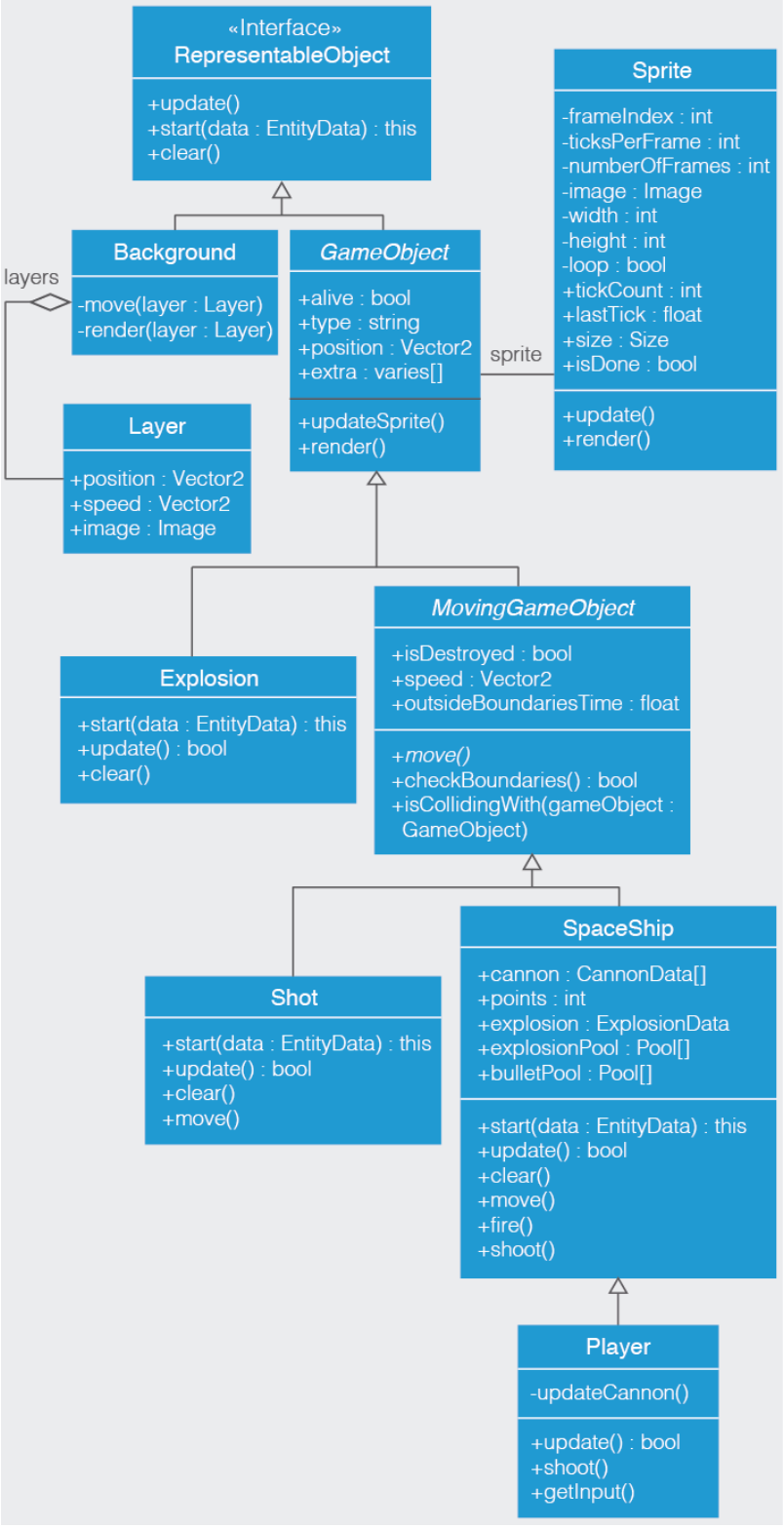
```
83
84      // Dades i funcions específiques de cada tipus d'enemic
85      that.extra = data.extra || {};
86
87      if (data.move) {
88        that.move = data.move.bind(that);
89      }
90
91      return that;
92    };
93
94    that.clear = function () {
95      that.isDestroyed = false;
96      that.alive = false;
97      that.type = null;
98      that.position = {x: 0, y: 0};
99      that.sprite = null;
100     that.speed = {x: 0, y: 0};
101     that.points = 0;
102     that.cannon = null;
103
104     // Dades i funcions específiques de cada tipus d'enemic
105     that.extra = null;
106     that.move = null;
107     that.outsideBoundariesTime = 0;
108   };
109
110   that.update = function () {
111
112     if (that.isDestroyed) {
113       that.explosionPool.instantiate(that.explosion, that.position);
114       return true;
115     }
116
117     that.updateSprite();
118     that.move();
119     that.fire();
120
121     if (that.checkBoundaries()) {
122       return true;
123     }
124
125     that.render();
126   };
127
128   return that;
129 };
130
131 var playerConstructor = function (options) {
132   var that = spaceshipConstructor(options);
133
134   function getInput() {
135     if (inputController.KEY_STATUS.left) {
136       that.position.x -= that.speed.x;
137     } else if (inputController.KEY_STATUS.right) {
138       that.position.x += that.speed.x;
139     }
140
141     if (inputController.KEY_STATUS.up) {
142       that.position.y -= that.speed.y;
143     } else if (inputController.KEY_STATUS.down) {
144       that.position.y += that.speed.y;
145     }
146
147     if (inputController.KEY_STATUS.space && !that.isDestroyed) {
148       that.fire();
149     }
150
151     // S'evita que surti de la pantalla
152     that.position.x = that.position.x.clamp(0, gameCanvas.width - that.sprite.
```

```
        size.width);
153     that.position.y = that.position.y.clamp(0, gameCanvas.height - that.sprite.
            size.height);
154
155   }
156
157   function updateCannon() {
158     for (var i = 0; i < that.cannon.length; i++) {
159       if (that.cannon[i].lastShot === undefined) {
160         that.cannon[i].lastShot = that.cannon[i].fireRate + 1;
161       }
162       that.cannon[i].lastShot++;
163     }
164   };
165
166   that.start(entitiesRepository.get('player', options.position, options.speed))
          ;
167
168   that.shoot = function (cannon) {
169     var origin;
170     if (cannon.lastShot > cannon.fireRate) {
171       cannon.lastShot = 0;
172       origin = {x: that.position.x + cannon.position.x, y: that.position.y +
              cannon.position.y};
173       that.bulletPool.instantiate(cannon.bullet, origin, cannon.direction);
174     }
175   };
176
177   that.update = function () {
178     updateCannon();
179
180     // Si ha sigut impactat, s'elimina. Aquí també es podria afegir l'animació
            de l'explosió
181     if (this.isDestroyed) {
182       that.explosionPool.instantiate(that.explosion, that.position);
183       return true;
184     }
185
186     getInput();
187     this.updateSprite();
188     this.render();
189   };
190
191   return that;
192 };
```

Com podeu veure, aquests objectes inclouen la utilització d'objectes de tipus `PoolGameObject` (`explosionPool` i `bulletPool`). Aquests objectes són un tipus especial de col·lecció que permet reduir la necessitat de crear objectes nous reutilitzant els que es troben al *pool* (conjunt d'objectes) en lloc de crear-ne de nous. Així, quan un objecte ja no és necessari (per exemple, una bala que surt de la pantalla o una nau que és destruïda), aquests objectes són marcats com a disponibles al *pool* i poden tornar a utilitzar-se quan calgui instanciar un nou objecte d'aquest tipus.

Fixeu-vos que la finalitat de cada objecte és molt clara tot i que aquesta jerarquia és força extensa: inclou classes abstractes i una interfície. A més a més, s'ha pogut evitar la duplicació de codi en fer les implementacions dels mètodes comuns a les classes pare o delegant-les a altres objectes (per exemple, tot el comportament relatiu als *sprites* és gestionat pels objectes `Sprite`).

Figura 1.28. Diagrama UML de la jerarquia completa de RepresentableObject



Cal destacar que quan a un diagrama apareix un mètode que ja es troba a una classe pare és perquè aquest mètode el sobreescriu (*override* en anglès). És a dir, quan s'invoca aquest mètode el codi executat serà el de la classe concreta on s'ha fet la invocació i no pas el de la classe pare.

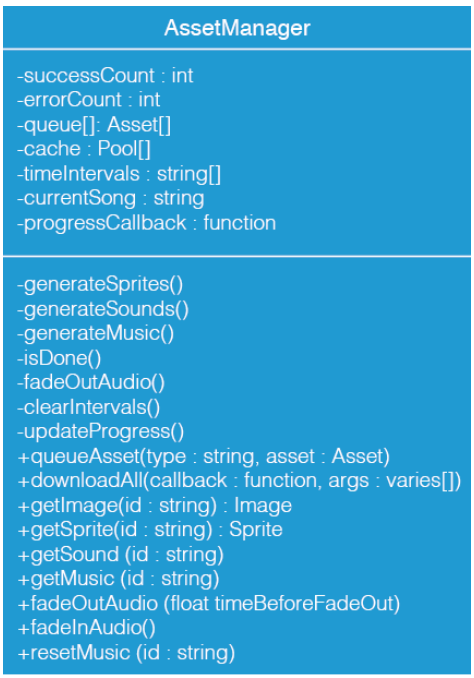
1.3.6 Gestor de recursos

Habitualment, un joc requerirà utilitzar imatges, *sprites*, efectes de so, música... A aquests elements se'ls coneix en conjunt com a recursos. Com que aquests recursos poden ser requerits per molts objectes diferents és recomanable utilitzar un objecte amb la funció de gestionar-los. Per exemple, un mateix efecte de so pot ser utilitzat per múltiples explosions al mateix temps i tots els enemics del mateix tipus faran servir el mateix *sprite*.

A JavaScript és especialment important fer-ho així, perquè s'han d'implementar mecanismes especials (com els gestors de descàrrega) per controlar quan s'han acabat de descarregar i inicialitzar tots els recursos, ja que la descàrrega es fa de manera asíncrona i l'execució del codi podria continuar abans d'haver descarregat totes les imatges per exemple.

A *IOC Invaders* la implementació del gestor de recursos (podeu veure el diagrama UML de l'`AssetManager` a la figura 1.29) inclou la gestió de descàrregues (només es controlen les imatges), la generació dels objectes necessaris, la funcionalitat d'obtenció de recursos i la reproducció d'àudio. Això s'ha fet així per simplificar el codi però és recomanable que cada objecte només tingui una responsabilitat.

FIGURA 1.29. Diagrama UML de la jerarquia completa de RepresentableObject



Com que en aquest objecte s'han barrejat múltiples funcionalitats, a continuació podeu trobar un resum dels mètodes i propietats dividits segons la seva utilitat:

- Gestió de descàrregues:
  - **queue**: llista d'elements per descarregar o generar.

  - **successCount** i **errorCount**: comptadors de recursos descarregats amb èxit o error.
  - **isDone**: mètode que determina si ja s'ha finalitzat la descàrrega dels recursos.
  - **progressCallback**: funció cridada per mostrar el progrés de la descàrrega.
  - **updateProgress**: mètode encarregat d'actualitzar el progrés de la descàrrega.
  - **queuesAsset**: mètode per afegir recursos a la llista de descàrrega.
  - **downloadAll**: mètode per iniciar la descàrrega dels recursos i engegar la generació dels objectes.

- Generació dels objectes:
  - **cache**: és la col·lecció d'objectes generats.
  - **generateSprites**: genera múltiples còpies de cada *sprite* per ser utilitzades com a *pool*.
  - **generateSounds**: genera múltiples còpies de cada efecte de so per generar un *pool* i poder reproduir-lo més d'una vegada simultàniament.
  - **generateMusic**: genera un element d'àudio per a cada objecte (només se'n necessita un en cada moment).

- Servei de recursos:
  - **getImage**: retorna la imatge indicada.
  - **getSprite**: retorna el primer *sprite* lliure del tipus indicat al *pool*.

- Reproducció d'àudio:
  - **currentSong**: identifica la música que es reprodueix en un moment concret.
  - **getSound**: reprodueix el primer so del tipus indicat que es trobi lliure al *pool* corresponent.
  - **getMusic**: reprodueix el fitxer de música indicat i el reinicia si ja estava reproduint-se.
  - **resetMusic**: atura la música i la reinicialitza.
  - **fadeInAudio**: abaixa el volum de tots els efectes de so a 0.
  - **fadeOutAudio**: restaura el volum de tots els efectes de so.
  - **timeIntervals**: són els temporitzadors utilitzats pel sistema de baixada i restauració del volum dels sons actius.
  - **clearInterval**: elimina els temporitzadors actius.

Com que l'única raó per la qual es "demana" un recurs d'àudio és per reproduir-lo, a *IOC Invaders* s'ha decidit reproduir-los automàticament en lloc de retornar-los quan s'invoquen els mètodes `getSound` i `getMusic`.

A continuació podeu trobar el codi complet del generador d'objectes de tipus `AssetManager`:

```
var assetManagerConstructor = function (progressCallback) {
    var that = {},
      successCount = 0,
      errorCount = 0,
      queue = {
        images: [],
        sprites: [],
        sounds: [],
        music: []
      },
      cache = {
        images: {},
        sprites: {},
        sounds: {},
        music: {}
      },
      timeIntervals = {},
      currentSong;

    function updateProgress() {
      if (progressCallback) {
        progressCallback(successCount + errorCount, queue.images.length);
      }
    }

    function generateSprites() {
      var pool, poolSize;

      for (var i = 0; i < queue.sprites.length; i++) {
        pool = [];
        poolSize = 10;

        for (var j = 0; j < poolSize; j++) {
          pool.push(spriteConstructor({
            image: that.getImage(queue.sprites[i].id),
            numberOfFrames: queue.sprites[i].numberOfFrames,
            ticksPerFrame: queue.sprites[i].ticksPerFrame,
            loop: queue.sprites[i].loop === undefined ? true : queue.sprites[i
                ].loop
          }));
        }

        cache.sprites[queue.sprites[i].id] = pool;
      }
    }

    function generateSounds() {
      var pool, poolSize, sound;

      for (var i = 0; i < queue.sounds.length; i++) {
        pool = [];
        poolSize = 10; // nombre màxim de sons idèntics que es reprodueixen al
            mateix temps
        for (var j = 0; j < poolSize; j++) {
          sound = new Audio(queue.sounds[i].path);
          sound.volume = queue.sounds[i].volume;
          pool.push(sound);
        }
        cache.sounds[queue.sounds[i].id] = {
          currentSound: 0,
          pool: pool,
          volume: queue.sounds[i].volume
        }
      }
    }

    function isDone() {
      return (queue.images.length == successCount + errorCount);
    }
```

```
68
69  function fadeOutAudio() {
70    for (var id in cache.sounds) {
71      for (var i = 0; i < cache.sounds[id].pool.length; i++) {
72        var sound = cache.sounds[id].pool[i];
73        sound.volume = 0;
74      }
75    }
76  }
77
78  function generateMusic() {
79    for (var i = 0; i < queue.music.length; i++) {
80      var sound = new Audio(queue.music[i].path);
81      sound.volume = queue.music[i].volume;
82      sound.loop = queue.music[i].loop;
83      cache.music[queue.music[i].id] = sound;
84    }
85  }
86
87  that.queueAsset = function (type, asset) {
88    queue[type].push(asset);
89  };
90
91  that.downloadAll = function (callback, args) {
92    if (queue.images.length === 0) {
93      callback();
94    }
95
96    for (var i = 0; i < queue.images.length; i++) {
97      var path = queue.images[i].path,
98        id = queue.images[i].id,
99        img = new Image();
100
101     img.addEventListener("load", function () {
102       successCount += 1;
103       updateProgress();
104
105       if (isDone()) {
106         generateSprites();
107         callback(args);
108       }
109     }, false);
110
111     img.addEventListener("error", function () {
112       errorCount += 1;
113       updateProgress();
114       if (isDone()) {
115         generateSprites();
116         callback(args);
117       }
118     }, false);
119
120     img.src = path;
121     cache.images[id] = img;
122   }
123
124   generateSounds();
125   generateMusic();
126 };
127
128 that.getImage = function (id) {
129   return cache.images[id];
130 };
131
132 that.getSprite = function (id) {
133   var pool = cache.sprites[id];
134   var sprite = pool[pool.length - 1];
135   pool.unshift(pool.pop());
136   return sprite;
137 };
```

```
138
139   that.getSound = function (id) {
140     var sounds = cache.sounds[id];
141
142     if (sounds.pool[sounds.currentSound].currentTime === 0
143       || sounds.pool[sounds.currentSound].ended) {
144         sounds.pool[sounds.currentSound].play();
145     }
146
147     sounds.currentSound = (sounds.currentSound + 1) % sounds.pool.length;
148   };
149
150   that.fadeOutAudio = function (timeBeforeFadeOut) {
151     that.clearIntervals();
152     if (timeBeforeFadeOut) {
153       timeIntervals.fadeOutAudio = setTimeout(fadeOutAudio, timeBeforeFadeOut);
154     } else {
155       fadeOutAudio();
156     }
157   };
158
159   that.fadeInAudio = function () {
160     that.clearIntervals();
161
162     for (var id in cache.sounds) {
163       for (var i = 0; i < cache.sounds[id].pool.length; i++) {
164         var sound = cache.sounds[id].pool[i];
165         sound.volume = cache.sounds[id].volume;
166       }
167     }
168   };
169
170   that.clearIntervals = function () {
171     clearInterval(timeIntervals.fadeInAudio);
172     clearInterval(timeIntervals.fadeOutAudio);
173   };
174
175   that.getMusic = function (id) {
176     that.resetMusic(currentSong);
177     cache.music[id].play();
178     currentSong = id;
179   };
180
181   that.resetMusic = function (id) {
182     if (!id) {
183       return;
184     }
185
186     if (!cache.music[id].ended) {
187       cache.music[id].pause();
188     }
189
190     if (cache.music[id].currentTime > 0) {
191       cache.music[id].currentTime = 0;
192     }
193   };
194
195   return that;
196 };
```

Podeu trobar més informació sobre el patró de disseny façana a l'enllaç següent: goo.gl/JU6JRh.

Habitualment, quan es necessita centralitzar diferents funcionalitats en un mateix objecte s'acostuma a aplicar el patró de disseny *façana* (*facade* en anglès). Aquest patró consisteix a crear una classe que exposa els mètodes públics necessaris però internament treballa amb múltiples classes diferents. D'aquesta manera el funcionament de la façana és transparent a l'usuari, com en aquest cas, que no ha de preocupar-se d'on prové realment un *sprite*, simplement invoca el mètode `getSprite` de l'`AssetManager` sense haver de saber-ne l'origen.

Cal destacar que aquest objecte utilitza un sistema de *pools* simplificat (es tracta d'un *array*), perquè no s'ha de controlar l'estat dels elements d'aquests *pools*.

1.3.7 Optimització: ús de 'pools'

Un dels objectius principals a l'hora de desenvolupar un joc és que el nombre de *frames* per segon sigui estable. Preferentment han de mantenir-se 60 *frames* per segon per aconseguir que el joc sigui fluid.

Cal tenir en compte que, a JavaScript, la destrucció dels objectes és gestionada pels navegadors, que de tant en tant inicien la recol·lecció de brossa per alliberar la memòria dels objectes que ja no s'estan fent servir. Això pot provocar caigudes crítiques en el nombre de *frames* per segon durant l'execució, i cal evitar-lo tant sí com no perquè durant l'execució del recol·lector de brossa l'aplicació pot quedar completament bloquejada.

Com que durant un joc en pocs minuts es poden crear un gran nombre d'objectes (per exemple les bales en un joc de trets), s'ha d'implementar un sistema que permeti reciclar aquests objectes. El més habitual és fer servir un *pool* (conjunt d'objectes).

Aquesta tècnica és força simple d'entendre i d'implementar. Consisteix a crear el màxim nombre d'objectes que s'hagin de mostrar en pantalla al mateix temps abans de començar el joc i emmagatzemar-los al *pool*. Llavors, en lloc de crear els objectes, s'agafen d'aquest *pool*, i quan no es necessiten es tornen a afegir al *pool*. Així es reciclen i no cal crear-ne més: això evita que es cridi el recol·lector de brossa sigui i que s'aturi l'execució del programa.

És a dir, si es requereix que a la pantalla hi hagi 500 bales actives es crearà un *pool* amb 500 bales. Quan calgui instanciar una bala, s'obté del *pool*, i una vegada surt de la pantalla o impacta en un adversari es retorna al *pool*.

Hi ha molts sistemes per determinar quin és el pròxim objecte a retornar, i depenent de les vostres necessitats podeu fer-ne servir un o altre. Per exemple, suposant que els objectes s'emmagatzemen en un *array* amb nom `pool`, es podria determinar de les maneres següents:

- Guardar en un comptador quin és el proper ítem a utilitzar (aquest és el sistema utilitzat al mètode `getSound` de l'`AssetManager`): `properIndex = (properIndex + 1) % pool.length`.
- Retornar sempre l'últim element i moure'l de l'última posició a la primera (aquest és el sistema utilitzat al mètode `getSprite` de l'`AssetManager` i l'objecte `GameObjectPool`), per exemple:

```
1 function getElement() {
2   var element = pool[pool.length – 1];
3   pool.unshift(pool.pop());
4   return element;
5 }
```

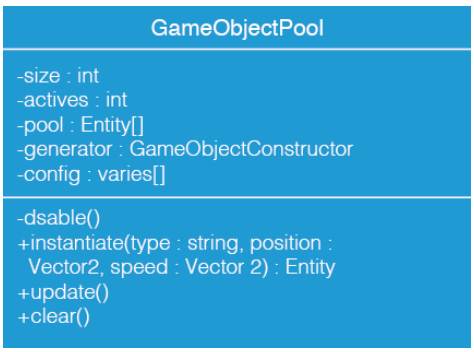
- Afegir una propietat a tots els objectes de tipus booleà `actiu`, canviar el valor a cert o fals (segons s'activi o es desactivi) i ordenar l'*array* segons aquesta propietat, de manera que tots els elements inactius es trobin al final de la llista, per exemple, i només calgui obtenir l'últim element, que sempre serà inactiu.
- Eliminar l'element de l'*array* (amb els mètodes `pop`, `shift` o `splice` segons la posició) quan sigui actiu i tornar a afegir-lo a l'*array* quan es desactivi.

L'objecte `GameObjectPool` del joc *IOC Invaders* combina dos sistemes (tots els mètodes pertanyen als *arrays*):

- En instanciar un objecte es mou de l'última posició (mètode `pop`) a la primera (mètode `unsfhit`).
- En desactivar un objecte s'elimina de la seva posició actual (mètode `splice`) i s'afegeix al final de l'*array* (mètode `push`).

El `GameObjectPool` permet netejar el *pool* (`clear`), instanciar nous objectes (`instantiate`) i actualitzar tots els `GameObjects` emmagatzemats (`update`), com es pot veure a la figura 1.30.

Figura 1.30. Diagrama UML objecte Pool



A continuació podeu trobar el codi del `GameObjectPool` implementat a l'*IOC Invaders*:

```
1  var gameObjectPoolConstructor = function (maxSize, generator, config) {
2    var that = {},
3    size = maxSize;
4
5    function disable(index) {
6      that.pool[index].clear();
7      that.pool.push((that.pool.splice(index, 1))[0]);
8    }
9
10   that.actives = size;
11   that.pool = [];
12
13   for (var i = 0; i < size; i++) {
14     that.pool[i] = generator(config);
15   }
16
17   that.instantiate = function (type, position, speed) {
```

```
18      var instance = that.pool[size - 1].start(entitiesRepository.get(type,
            position, speed));
19      that.pool.unshift(that.pool.pop());
20      return instance;
21    };
22
23    that.update = function () {
24      for (var i = 0; i < size; i++) {
25        // Només dibuixem fins que trobem un objecte que no sigui viu
26        if (that.pool[i].alive) {
27          if (that.pool[i].update()) {
28            // Si update ha retornat cert, és que s'ha de desactivar
29            disable(i);
30          }
31        } else {
32          that.actives = i;
33          break;
34        }
35      }
36    };
37
38    that.clear = function () {
39      for (var i = 0; i < size; i++) {
40        that.pool[i].alive = false;
41      }
42      that.actives = 0;
43    };
44
45    return that;
46  };
```

Com es pot apreciar, el generador d'objectes rep com a paràmetres `maxSize` (mida del *pool*), `generator` (generador d'objectes) i `config` (dades de configuració per generar els objectes). És a dir, es pot passar com a generador la funció `shotConstructor` per crear un *pool* de bales o la funció `spaceshipConstructor` per crear-ne un de naus enemigues.

La funció `clear` permet netejar el *pool* quan es reinicia el joc o es passa de nivell, mentre que la funció `update` és cridada des del bucle principal del joc a cada iteració i s'encarrega d'actualitzar cadascun dels elements del *pool*.

Fixeu-vos que en el cas de `update` només es recorren els elements fins a trobar-ne un que no estigui actiu (corresponent a la propietat `alive`), ja que els elements es troben ordenats i no cal continuar amb el bucle.

1.3.8 Motor del joc

Quan es parla del **motor del joc** cal distingir entre el motor implementat concretament per a un joc (per exemple el motor del *IOC Invaders*), i els motors de joc genèrics utilitzats per desenvolupar jocs (com Construct 2 o Game Canvas). En aquesta secció ens referim als primers, la implementació de motors de joc propis.

El motor del joc és el component responsable de portar a terme les tasques genèriques del joc, com poden ser, entre d'altres, les següents:

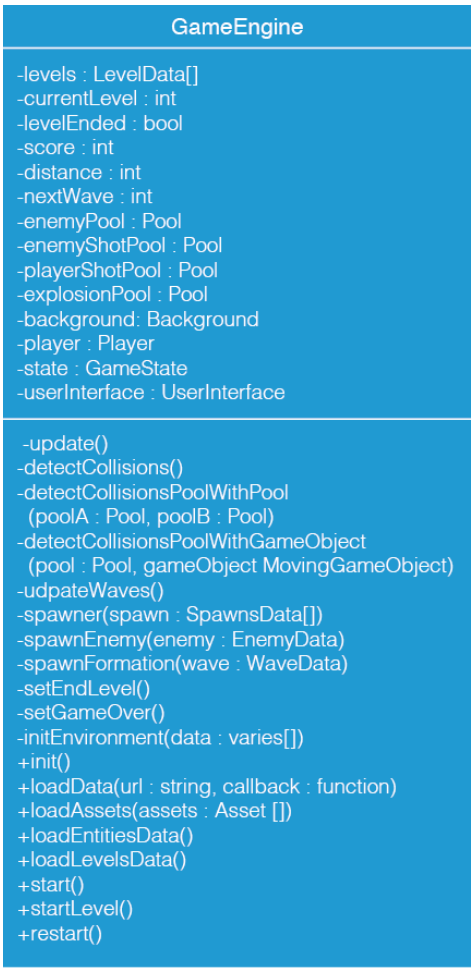
- Inicialitzar el joc.

- Gestionar la detecció de col·lisions.
- Carregar les dades del joc.
- Gestionar els canvis de pantalles.
- Executar el bucle principal del joc.

L'execució del bucle principal és especialment important, perquè una vegada el joc estigui inicialitzat, aquest bucle es repetirà a cada *frame* i serà el responsable d'actualitzar tots els elements del joc.

Com es pot apreciar a la figura 1.31 el motor del joc *IOC Invaders* (`GameManager`) es força complex, ja que inclou múltiples funcionalitats en un sol objecte.

FIGURA 1.31. Diagrama UML objecte GameManager

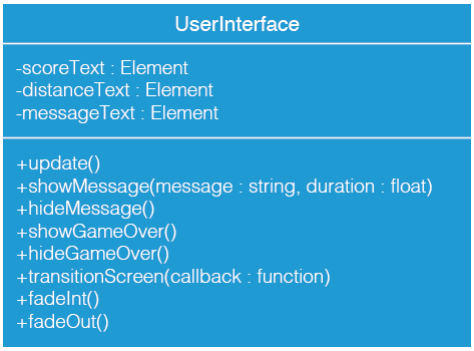


Interfície d'usuari

La interfície d'usuari del joc és l'encarregada de mostrar informació a l'usuari i permet interactuar-hi mostrant un menú d'opcions per iniciar el joc o les puntuacions. També es consideren elements de la interfície els indicadors que es poden fer servir per mostrar els punts de vida d'un enemic o un ressaltat al voltant d'un tresor.

Per simplificar-ho, a *IOC Invaders* només es mostra la puntuació, la distància recorreguda al nivell i un botó per reiniciar el joc quan acaba la partida. Com que es tracta d'un cas tan simple, s'ha optat per afegir l'objecte `UserInterface` (vegeu la figura 1.32) directament a l'objecte `GameEngine`, però no és el més recomanable.

Figura 1.32. Diagrama UML objecte UserInterface



A continuació podeu trobar un resum dels mètodes i la seva funció:

- **update**: actualitza la puntuació i la distància en pantalla.
- **showMessage** i **hideMessage**: mostra i amaga missatges respectivament, per exemple, en començar un nivell.
- **showGameOver** i **hideGameOver**: mostra i amaga el missatge de fi del joc (inclou el botó per tornar a començar mitjançant HTML).
- **transitionScreen**, **fadeIn**, **fadeOut**: fan efectes per enfosquir i tornar a mostrar la pantalla.

Càrrega de dades

Cal diferenciar entre la càrrega de dades i la càrrega de recursos. Normalment els recursos es corresponen amb elements d'HTML (àudio, vídeo i imatges) i es poden controlar quan ha finalitzat la càrrega detectant l'*event* `load`, ja que es tracta d'elements. En canvi, les dades s'han de carregar realitzant peticions AJAX i processant-ne la resposta.

En cas que els fitxers amb les dades es trobin en un servidor diferent, s'ha de tenir en compte que aquest ha d'implementar mecanismes CORS o JSONP per evitar el bloqueig dels navegadors provocat per la política del mateix origen.

Al motor del joc de l'*IOC Invaders* s'utilitzen les següents funcions per portar a terme la càrrega de dades:

- **loadData**: realitza la petició AJAX i invoca la funció passada com a argument per processar les dades rebudes en format JSON.
- **loadEntitiesData**: carrega les dades de les entitats i les afegeix al repositori.

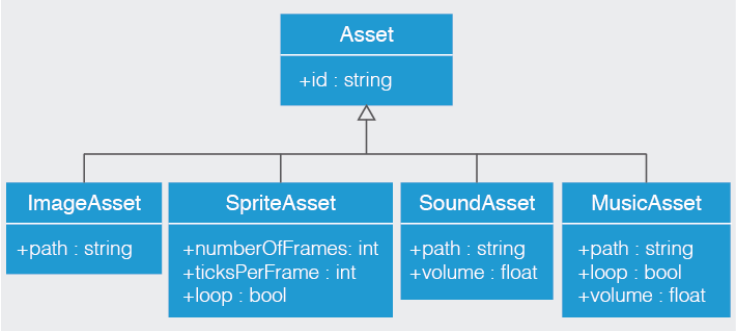
- **loadLevelsData**: carrega les dades del nivell i, en acabar, inicia el joc.
- **init** i **loadAssets**: el primer carrega les dades dels recursos i seguidament n'inicia la descàrrega mitjançant l' `AssetsManager`.

Totes les dades del joc són en format JSON, cosa que facilita la tasca d'editar-los manualment o crear un editor per modificar-los.

A continuació podeu trobar l'estructura del fitxer de recursos (`asset-data.json`) que correspon al diagrama de la figura 1.33:

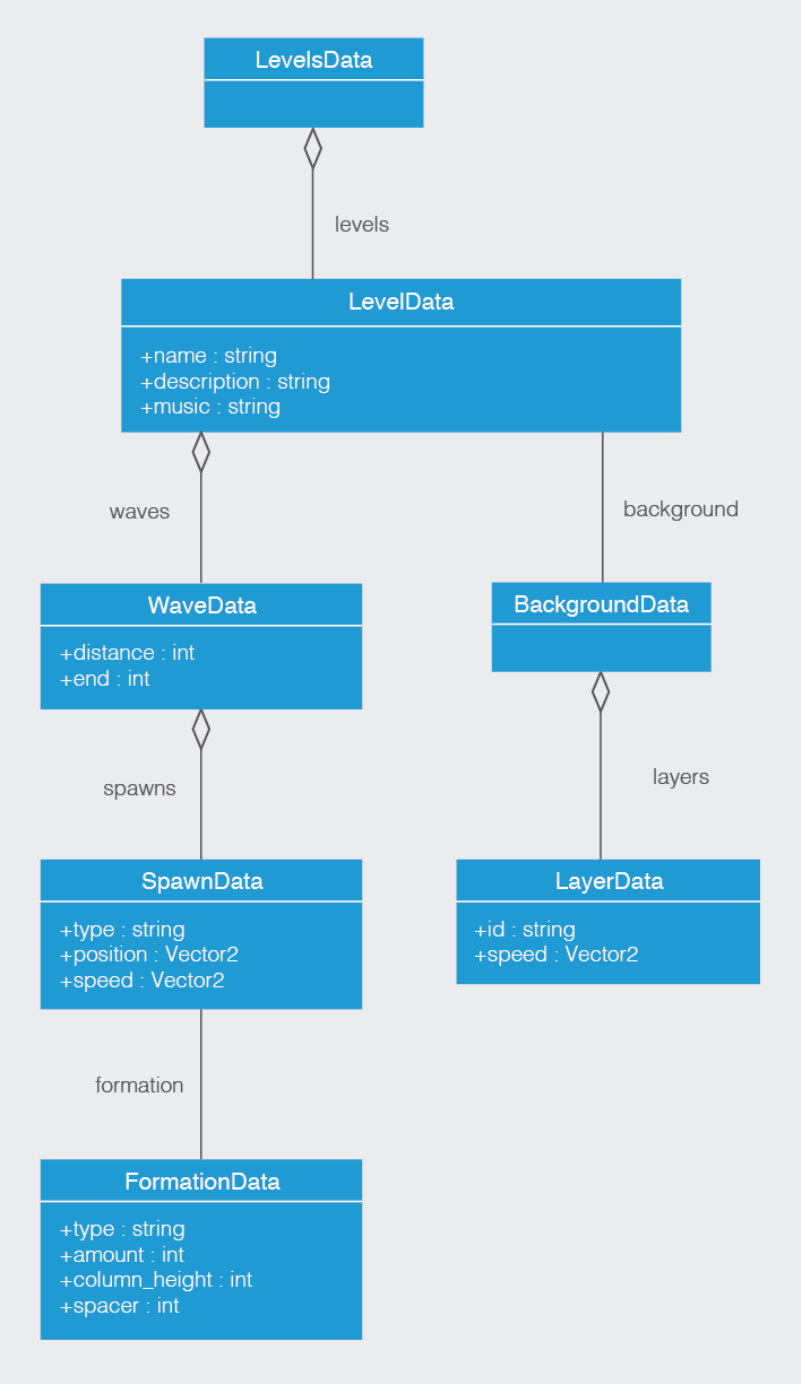
```
{
  "assets": {
      "images": [
        {
          "id": "identificador de la imatge",
          "path": "ruta de la imatge"
        }
      ],
      "sprites": [
        {
          "id": "identificador del sprite",
          "numberOfFrames": "nombre de frames que formen l'animació",
          "ticksPerFrame": "nombre de frames que es mostren a la mateixa imatge
               abans de canviar a la següent"
        }
      ],
      "sounds": [
        {
          "id": "identificador de l'efecte de so",
          "path": "ruta de l'efecte de so",
          "volume": "tipus float. Volumen al qual s'ha de reproduir"
        }
      ],
      "music": [
        {
          "id": "identificador del fitxer de música",
          "path": "ruta del fitxer de música",
          "loop": "tipus booleà. Indica si s'ha de repetir indefinidament"
          "volume": "tipus float. volumen al que s'ha de reproduir"
        }
      ]
  }
}
```

Figura 1.33. Estructura de dades de "asset-data.json"



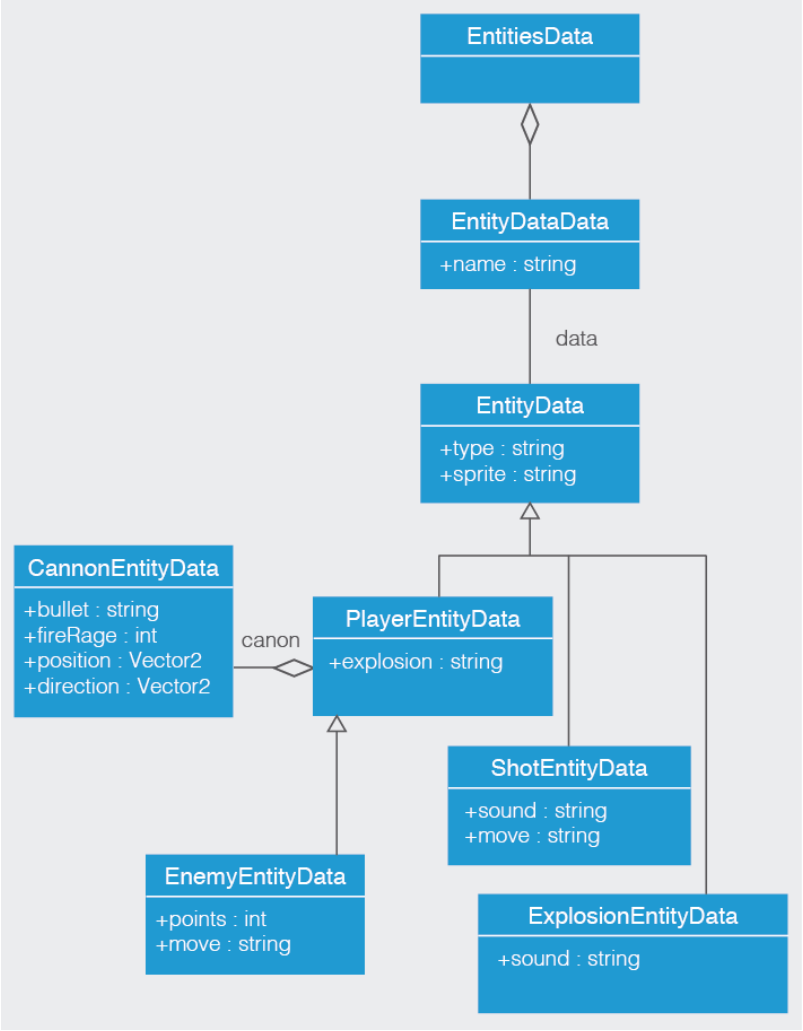
L'estructura dels fitxers `levels-data.json` és més complicada perquè conté múltiples estructures niudes (nivell, fons, onades d'enemics i formacions), però a la figura 1.34 se'n pot veure clarament la composició.

Figura 1.34. Estructura de dades de "levels-data.json"



El fitxer `entity-data.json` també inclou estructures niades. En aquest cas el fitxer inclou la informació del jugador, dels diferents tipus d'enemics i el seu armament, i les bales i les explosions, com es pot apreciar a la figura 1.35.

FIGURA 1.35. Estructura de dades de "entity-data.json"



Fixeu-vos que tot i que el més habitual és que cada nau només porti un canó, és possible afegir-hi qualsevol quantitat, ja que la propietat `cannon` és un *array*. Així, per exemple, la nau del jugador inclou dos canons col·locats a les posicions (35,5) i (35,45).

Com que la bala del jugador té una alçada de 14 píxels, l'alçada de la nau és de 64 píxels i l'origen de les coordenades (0,0) es troba a la cantonada superior esquerra. Per col·locar el segon canó simètric al primer s'ha de fer el càlcul següent: `64 - 14 - 5 = 45`. Per la distància horitzontal s'utilitza el mateix valor a tots dos canons.

Detecció de col·lisions

La detecció de col·lisions s'ha de calcular a cada iteració del bucle principal. Les comprovacions seran més o menys costoses segons la forma dels elements per comprovar. Per exemple, les dues formes de detectar col·lisions més simples són:

- **Forma circular**: requereix calcular si la distància que hi ha entre el centre de tots dos cercles és més gran que la suma dels radis.

- **Rectangles**: cal comprovar si qualsevol de les cantonades del primer rectangle es troba dintre de l'àrea del segon.

Els motors de joc de tercers permeten utilitzar formes més avançades en 2D i 3D per detectar col·lisions.

En cas que es necessités més precisió en la detecció, es poden utilitzar múltiples cercles i rectangles per ajustar la forma utilitzada per fer els càlculs, però –com es pot intuir– el cost de realitzar-los es multiplica.

A *IOC Invaders* la detecció de col·lisions està dividida entre dos components i s'utilitza la detecció de col·lisions per rectangles. Per una banda, els objectes de tipus `MovingGameObject`, com les naus i les bales, implementen `isCollidingWith`, que permet determinar si un objecte està col·lidint amb un altre. Per l'altra, en el motor del joc es troben els mètodes següents:

- **detectCollisions**: comprova totes les col·lisions possibles i marca com a destruïts els elements que hi hagin col·lidit.
- **detectCollisionsPoolWithPool**: detecta totes les col·lisions entre dos *pools* complets, per exemple el *pool* de bales del jugador amb el *pool* de naus enemigues.
- **detectCollisionsPoolWithGameObject**: detecta les col·lisions entre tots els objectes d'un *pool* amb un objecte concret, per exemple, el *pool* de bales enemigues amb el jugador.

1.3.9 Gestió d'enemics i nivells

El motor del joc *IOC Invaders* també és el responsable d'instanciar els enemics a mesura que el jugador arriba als punts marcats (propietat `distance`). Els mètodes responsables de generar els enemics són els següents:

- **updateWaves**: comprova si s'ha arribat a la distància requerida per instanciar un nou grup d'enemics o s'ha arribat al final del nivell.
- **spawner**: instancia un nou enemic o una nova onada d'enemics.
- **spawnEnemy**: instancia un únic enemic.
- **spawnFormation**: instancia una onada d'enemics en formació. Segons el tipus de formació i les dades de l'onada, les naus es col·locaran en diferents posicions.

S'ha de tenir en compte que, per simplificar, aquesta funcionalitat s'ha afegit al `GameManager`, però el més adient seria crear un objecte de tipus `SpawnManager` que fos l'encarregat de gestionar la creació d'instàncies dels enemics.

Aquests enemics, en ser destruïts, augmenten la puntuació del jugador. La puntuació ha de ser gestionada pel `GameManager`, ja que a diferència d'altres elements, la puntuació ha de persistir durant tota la partida.

Quant a la gestió de nivells, a *IOC Invaders* s'utilitzen les propietats i els mètodes següents:

- **currentLevel**: indica el nivell actual.
- **levels**: informa sobre els nivells del joc.
- **startLevel**: inicia el nivell.
- **setEndLevel**: avança al següent nivell. És invocat quan s'arriba al final del nivell i no és visible cap enemic.

Inicialització i bucle principal del joc

La resta de mètodes del `GameEngine` s'encarreguen de gestionar el cicle de vida del joc. És a dir, la inicialització, la finalització i el bucle principal:

- **start**: és invocat en iniciar-se el joc, posa en marxa el bucle principal (`update`).
- **restart**: invocat tant quan s'inicia el joc com quan es reinicia.
- **init**: inicia la càrrega de dades i seguidament invoca `initEnvironment`.
- **initEnvironment**: inicialitza els *pools* per les bales, les explosions i els enemics, estableix el context del canvas (`2D`) i inicia la càrrega de recursos.
- **setGameOver**: atura el funcionament del joc i mostra el missatge de fi del joc.
- **update**: executa el bucle principal del joc.

Quan es treballa amb l'element `canvas` és fonamental invocar el mètode `window.requestAnimationFrame`, ja que és l'encarregat de demanar al navegador que es cridi la funció passada com a argument abans de pintar la finestra.

Com es pot apreciar en el bucle principal de l'*IOC Invaders* (`update`), la primera acció que es porta a terme és invocar `windows.requestAnimationFrame` per tornar a ser cridat quan es torni a pintar la finestra. Seguidament s'actualitzen les onades i es detecta si s'ha produït alguna col·lisió.

Cal destacar que s'utilitza una màquina d'estats molt simple per determinar què s'ha de fer a continuació:

- Si el jugador ha estat destruït i l'estat no és `GAME_OVER`, s'acaba el joc.
- Si no hi ha enemics vius, s'ha arribat al final del nivell i l'estat no és `GAME_OVER`, l'estat passa a `LOADING_NEXT_LEVEL` i s'inicia la transició cap al pròxim nivell.
- Si no s'ha acabat el nivell i no s'ha acabat el joc, s'actualitza el jugador (estat `RUNNING`).

Podeu trobar més informació sobre les màquines d'estats finits a l'enllaç següent: goo.gl/t0j8OA.

Abans de finalitzar el bucle s'actualitza el *pool* d'explosions i finalment la interfície amb la nova distància i la puntuació.

Fixeu-vos que principalment la tasca del bucle principal és cridar el mètode `update` dels altres elements del joc i canviar d'estat. Si estructureu bé el vostre codi, la implementació del bucle principal és molt simple.

A continuació podeu trobar el codi del generador d'objectes de tipus `GameEngine`:

```
var gameEngineConstructor = function () {
  var that = {},
  levels = {},
  currentLevel,
  levelEnded,
  score,
  distance, // Relativa al nivell actual
  nextWave, // Relativa al nivell actual

  enemyPool,
  enemyShotPool,
  playerShotPool,
  explosionPool,

  background,
  player,
  state,

  ui = (function () {
    var scoreText = document.getElementById('score'),
      distanceText = document.getElementById('distance'),
      messageText = document.getElementById('messages');

    return {
      update: function () {
        scoreText.innerHTML = score;
        distanceText.innerHTML = distance;
      },

      showMessage: function (message, duration) { // Temps en mil·lisegons
        messageText.innerHTML = message;
        messageText.style.opacity = 1;

        setTimeout(function () {
          messageText.style.opacity = 0;
        }, duration);
      },

      hideMessage: function () {
        messageText.style.opacity = 0;
      },

      showGameOver: function () {
        document.getElementById('game-over').style.display = "block";
      },

      hideGameOver: function () {
        document.getElementById('game-over').style.display = "none";
      },

      transitionScreen: function (callback) {
        gameCanvas.style.opacity = 0;

        setTimeout(function () {
          gameCanvas.style.opacity = 1;
          callback();
        }, 3000); // la transició dura 3s
      },

```

```
60        fadeIn: function () {
61          gameCanvas.style.opacity = 1;
62        },
63
64        fadeOut: function () {
65          gameCanvas.style.opacity = 0;
66        }
67      };
68    })();
69
70    function initEnvironment(data) {
71      gameContext = gameCanvas.getContext("2d");
72
73      explosionPool = gameObjectPoolConstructor(100, explosionConstructor);
74      enemyShotPool = gameObjectPoolConstructor(500, shotConstructor);
75      enemyPool = gameObjectPoolConstructor(100, spaceshipConstructor, {
76        pool: {
77          bullet: enemyShotPool,
78          explosion: explosionPool
79        }
80      });
81
82      playerShotPool = gameObjectPoolConstructor(100, shotConstructor);
83
84      that.loadAssets(data.assets);
85    }
86
87    function update() {
88      window.requestAnimationFrame(update);
89
90      updatedSprites = [];
91
92      updateWaves();
93      detectCollisions();
94
95      background.update();
96      enemyPool.update();
97      enemyShotPool.update();
98      playerShotPool.update();
99
100
101     if (player.isDestroyed && state != GameState.GAME_OVER) {
102       setGameOver();
103
104     } else if (enemyPool.actives === 0 && levelEnded && state != GameState.
            GAME_OVER
105         && state != GameState.LOADING_NEXT_LEVEL) {
106       ui.transitionScreen(setEndLevel)
107       state = GameState.LOADING_NEXT_LEVEL;
108     } else if (state != GameState.GAME_OVER) {
109       player.update();
110       distance++;
111     }
112
113     explosionPool.update();
114
115     ui.update();
116   }
117
118   function detectCollisions() {
119     var impactInfo,
120       i;
121
122     // bala del jugador amb enemic
123     impactInfo = detectCollisionsPoolWithPool(playerShotPool, enemyPool);
124
125     if (impactInfo.length > 0) {
126       for (i = 0; i < impactInfo.length; i++) {
127         impactInfo[i].source.isDestroyed = true;
128         impactInfo[i].target.isDestroyed = true;
```

```
        score += impactInfo[i].target.points;
      }
    }

    // bala del enemic amb jugador
    impactInfo = detectCollisionsPoolWithGameObject(enemyShotPool, player);

    if (impactInfo.length > 0) {
      for (i = 0; i < impactInfo.length; i++) {
        impactInfo[i].source.isDestroyed = true;
        impactInfo[i].target.isDestroyed = true;
      }
    }

    // enemic amb jugador
    impactInfo = detectCollisionsPoolWithGameObject(enemyPool, player);
    if (impactInfo.length > 0) {
      for (i = 0; i < impactInfo.length; i++) {
        impactInfo[i].source.isDestroyed = true;
        impactInfo[i].target.isDestroyed = true;
      }
    }
  }

  function detectCollisionsPoolWithPool(poolA, poolB) {
    var i = 0,
      j = 0,
      impacts = [];

    while (poolA.pool[i] && poolA.pool[i].alive) {
      while (poolB.pool[j] && poolB.pool[j].alive) {
        if (poolA.pool[i].isCollidingWith(poolB.pool[j])) {
          impacts.push({
            source: poolA.pool[i],
            target: poolB.pool[j]
          });
        }
        j++;
      }
      j = 0;
      i++;
    }

    return impacts;
  }

  function detectCollisionsPoolWithGameObject(pool, gameObject) {
    var i = 0,
      impacts = [];

    while (pool.pool[i] && pool.pool[i].alive) {
      if (pool.pool[i].isCollidingWith(gameObject)) {
        impacts.push({
          source: pool.pool[i],
          target: gameObject
        });
      }
      i++;
    }

    return impacts;
  }

  function updateWaves() {
    var waves = levels[currentLevel].waves,
      currentWave;

    if (nextWave < waves.length && distance >= waves[nextWave].distance) {
      currentWave = waves[nextWave];
      spawner(currentWave.spawns);
```

```
      nextWave++;
    }

    if (distance > levels[currentLevel].end) {
      levelEnded = true;
    }
  }

  function spawner(spawns) {
    for (var i = 0; i < spawns.length; i++) {
      if (!spawns[i].formation) {
        spawnEnemy(spawns[i]);
      } else {
        spawnFormation(spawns[i]);
      }
    }
  }

  function spawnEnemy(enemy) {
    enemyPool.instantiate(enemy.type, enemy.position, enemy.speed);
  }

  function spawnFormation(wave) {
    var i,
      j,
      originPosition,
      spacer,
      nextColumn,
      currentPosition,
      side;

    switch (wave.formation.type) {
      case "columns":
        originPosition = {x: wave.position.x, y: wave.position.y};
        spacer = wave.formation.spacer;
        nextColumn = originPosition.y + wave.formation.column_height;
        currentPosition = {x: wave.position.x, y: wave.position.y};

        for (i = 0; i < wave.formation.amount; i++) {
          enemyPool.instantiate(wave.type, {
            x: currentPosition.x,
            y: currentPosition.y
          }, wave.speed);

          currentPosition.y += spacer;

          if (currentPosition.y >= nextColumn) {
            currentPosition.x += spacer;
            currentPosition.y = originPosition.y;
          }
        }
        break;

      case "grid":
        originPosition = {x: wave.position.x, y: wave.position.y};
        spacer = wave.formation.spacer;
        side = Math.round(Math.sqrt(wave.formation.amount));

        for (i = 0; i < side; i++) {
          for (j = 0; j < side; j++) {
            enemyPool.instantiate(wave.type, {
              x: originPosition.x + (spacer * i),
              y: originPosition.y + (spacer * j)
            }, wave.speed);
          }
        }

        break;

      case "row":
```

```
269         originPosition = {x: wave.position.x, y: wave.position.y};
270         spacer = wave.formation.spacer;
271
272         for (i = 0; i < wave.formation.amount; i++) {
273           enemyPool.instantiate(wave.type, {
274             x: originPosition.x + (spacer * i),
275             y: originPosition.y
276           }, wave.speed);
277         }
278         break;
279
280       case "column":
281         originPosition = {x: wave.position.x, y: wave.position.y};
282         spacer = wave.formation.spacer;
283
284         for (i = 0; i < wave.formation.amount; i++) {
285           enemyPool.instantiate(wave.type, {
286             x: originPosition.x,
287             y: originPosition.y + (spacer * i)
288           }, wave.speed);
289         }
290         break;
291
292       default:
293         console.log("Error, no es reconeix el tipus de formació");
294     }
295   }
296
297   function setEndLevel() {
298     currentLevel++;
299
300     if (currentLevel >= levels.length) {
301       currentLevel = 0;
302     }
303
304     enemyPool.clear();
305     explosionPool.clear();
306     enemyShotPool.clear();
307     playerShotPool.clear();
308
309     that.startLevel(currentLevel);
310     state = GameState.RUNNING;
311   }
312
313
314   function setGameOver() {
315     player.update();
316     state = GameState.GAME_OVER;
317
318     ui.hideMessage();
319     ui.showGameOver();
320     ui.fadeOut();
321     assetManager.fadeOutAudio(2000); // Donem temps perquè es produeixi l'
          explosió
322     setTimeout(function () {
323       assetManager.getMusic("game_over");
324     }, 2000);
325   }
326
327   that.init = function () {
328     that.loadData(config.asset_data_url, initEnvironment);
329   };
330
331   that.loadData = function (url, callback) {
332     var httpRequest = new XMLHttpRequest();
333
334     httpRequest.open("GET", url, true);
335     httpRequest.overrideMimeType('text/plain');
336     httpRequest.send(null);
337
```

```
    httpRequest.onload = function () {
      var data = JSON.parse(httpRequest.responseText);
      callback(data);
    };
  };

  that.loadAssets = function (assets) {
    for (var type in assets) {
      for (var i = 0; i < assets[type].length; i++) {
        assetManager.queueAsset(type, assets[type][i]);
      }
    }

    assetManager.downloadAll(that.loadEntititesData);
  };

  that.loadEntititesData = function () {
    that.loadData(config.entity_data_url, function (data) {
      entitiesRepository.add(data);
      that.loadLevelsData();
    })
  };

  that.loadLevelsData = function () {
    that.loadData(config.levels_data_url, function (data) {
      levels = data.levels;
      that.start();
    });
  };

  that.start = function () {
    background = backgroundConstructor({
      context: gameContext
    });

    that.restart();
    update();
  };

  that.startLevel = function (level) {
    var message = levels[level].name
      + "<p><span>"
      + levels[level].description
      + "</span></p>";

    ui.showMessage(message, 3000);

    background.start(levels[level].background);
    assetManager.getMusic(levels[currentLevel].music);

    levelEnded = false;
    nextWave = 0;
    distance = 0;

    player.position = {x: 10, y: 256};
  };

  that.restart = function () {
    ui.hideGameOver();
    assetManager.fadeInAudio();

    player = playerConstructor(
      {
        pool: {
          bullet: playerShotPool,
          explosion: explosionPool
        },
        position: {x: 10, y: 256},
        speed: {x: 4, y: 4}
      });
```

```
408
409     currentLevel = 0;
410     score = 0;
411     state = GameState.RUNNING;
412
413     enemyPool.clear();
414     explosionPool.clear();
415     enemyShotPool.clear();
416     playerShotPool.clear();
417
418     assetManager.resetMusic(levels[currentLevel].music);
419
420     ui.fadeIn();
421     that.startLevel(currentLevel);
422   };
423
424   return that;
425 };
```

Recordeu que podeu trobar el codi complet del joc *IOC Invaders* a l'enllaç següent: github.com/XavierGaro/ioc-invaders.