ASTEROIDS AMB JAVASCRIPT

Agraïments

Aquest document, la presentació HTML i la xerrada feta a les jornades tècniques de l'Institut de l'Ebre 2016 es deuen a l'obstinació del meu fill, que va voler programar un videojoc pel seu treball de recerca de 2^{n} de BAT. Va fer una versió de l'Asteroids amb Python, i al demanar-me si era possible posar-la en un navegador vaig començar a cercar informació sobre els jocs en un entorn Web. Jo havia treballat durant molts anys com a programador en l'entorn empresarial, però mai havia tingut cap contacte amb la programació de videojocs. Cercar una forma de mostrar un joc en el navegador em va dur a conèixer el framework PHASER i a pensar que potser seria interessant pels alumnes dels cicles d'informàtica aproximar-se a la programació a partir dels jocs. El codi d'aquest joc és una adaptació del què el meu fill va desenvolupar pel seu treball de recerca. Com a pare resulta molt gratificant poder aprendre del teu fill.

Una xerrada amb Sergi Tur, amb qui sempre aprenc coses noves, em va acostar a una llibreria[1] per controlar el moviment en dispositius tàctils. No he reutilitzat la llibreria, però algunes de les idees d'aquell treball es troben en el controlador tàctil del joc desenvolupat.

El meu agraïment a tots dos, així com als desenvolupadors de PHASER, és un framework molt complet, però sobre tot amb una documentació excel·lent i un fòrum on hi ha resolts un bon nombre de dubtes i d'una forma molt pedagògica.

Presentació

Aquest document vol ser un recurs pels que vulguin seguir la xerrada de les jornades tècniques, però també un petit tutorial pels que vulguin començar a aprendre a desenvolupar jocs amb JavaScript fent servir el framework PHASER.

Comentaré alguns dels principals aspectes relacionats amb el desenvolupament web i després explicaré en una seqüència de passos com desenvolupar una versió del joc arcade Asteroids.

Tot el codi del joc, dels diferents passos proposat, així com la presentació feta en les

1 http://seb.ly/2011/04/multi-touch-game-controller-in-javascripthtml5-for-ipad/

CC BY

jornades tècniques, i aquest mateix document, es troben a gitHub, en l'enllaç: https://github.com/jaumeramos/asteroids És un repositori públic, així que podeu fer correccions o proposar millores i solucions alternatives al codi proposat.

La presentació feta per les jornades tècniques és una pàgina de GitHub-Pages i també la podeu veure a: http://jaumeramos.github.io/asteroids/ dins al repositori GitHub es troba a la branca «gh-pages».

Al final d'aquest document teniu informació d'algunes eines que podeu utiiltzar per desenvolupar. Són les que jo he fet servir, i teniu instruccions per instal·lar-les en Linux (Debian, Ubuntu, Mint...).

CC BY

Índex de continguts

Desenvolupament de jocs amb JavaScript

Per què desenvolupem jocs amb JavaScript? En una xerrada a MediterráneaJS sobre Phaser i JavaScript, Belen Albeza[2] donava una raó contundent: «*Because we can :)*». Certament fa no molts anys era totalment inviable plantejar-se el desenvolupament de jocs en un entorn Web, i les alternatives que hi havia eren utilitzant extensions o plugins propietaris com per exemple Flash.

De fet, quan Sir Tim Berners-Lee va desenvolupar HTTP i HTML l'any 1989, poc es podria haver imaginat que un protocol pensat per poder accedir a fitxers de text fent servir enllaços hipertextuals, acabaria esdevenint un dels principals interfícies pels usuaris de ordinadors, tablets, dispositius mòbils, etc. Permetent dur a terme tasques com l'edició de textos, el desenvolupament de programes, el retoc d'imatges, i un llarg etcètera. De fet avui en dia trobem a la Web aplicacions per fer pràcticament qualsevol cosa que puguem necessitar.

Aquesta evolució de la Web ha tingut dos línies diferents, per una banda s'han desenvolupat alternatives per poder mostrar contingut dinàmic des dels servidors. És a dir, que enlloc de tornar fitxers HTML estàtics que algú havia creat, els servidors ara creen de forma dinàmica el contingut de les pàgines web, a partir d'informació que es troba en bases de dades o fins i tot en altres servidors Web. Per altra banda també s'ha desenvolupat molt el processament de la part del client, és a dir del navegador. Aquesta línia ha estat liderada principalment per JavaScript, un llenguatge creat per Brendan Eich que aparegué per primer cop al navegador Netscape cap a l'any 1995. En un principi fou utilitzat per oferir funcionalitats bàsiques per crear scripts senzills al navegador, i que ha anat evolucionant fins convertir-se en un llenguatge molt potent que es troba darrera de la majoria d'interfícies Web.

El desenvolupament de JavaScript ha seguit diferents línies, per una part el propi llenguatge, que ha passat de tenir diferents implementacions molt limitades i incompatibles entre elles a esdevenir un llenguatge molt potent i ràpid i molt més estandaritzat, de forma que les implementacions dels diferents navegadors ja no són incompatibles, facilitant molt

2 http://www.belenalbeza.com/talk-at-mediterraneajs-2015/

CC BY

la feina dels desenvolupadors. La versió més utilitzada actualment de JavaScript s'anomena EcmaScript 5, és un estàndard aprovat per la ECMA (European Computer Manufacturers Association) al 2009. El juny del 2015 s'aprovà la versió 6 de EcmaScript, anomenada EC6 o Harmony, ja que és un estàndard que ha estat desenvolupat de forma conjunta pels principals actors en el desenvolupament de la Web. Aquest nou estàndard encara no està implementat de forma generalitzada en tots els navegadors. EC6 aporta moltes novetats al llenguatge JavaScript, per tal de millorar tant la seva potència com per simplificar la codificació dels programes.

Un altra línia de desenvolupament de JavaScript ha estat la normalització del estàndard DOM (Document Object Model) que permet accedir d'una forma jeràrquica a tots els elements d'una pàgina HTML de forma que es pot modificar tant el seu contingut com el seu aspecte. Aquest model va ser creat per JavaScript, i degut a les incompatibilitats entre implementacions en els diferents navegadors va acabar sent estandaritzat pel W3C a partir de1998. Aquest model, juntament amb la tecnologia AJAX (Asynchronous JavaScript And XML) que permet amb JavaScript fer crides asíncrones (no hem d'esperar una resposta per seguir) al servidor Web ha millorat molt la usabilitat de l'entorn Web, tot i seguir fent servir un protocol com HTTP que no va estar dissenyat en cap moment pensant en els usos que té avui en dia.

La darrera línia que ha complementat l'evolució de JavaScript ha estat HTML5. Aquesta darrera versió de l'estàndard HTML aporta moltes possibilitats a l'entorn web, especialment el fet de poder dibuixar amb JavaScript elements gràfics directament al navegador amb el nou element Canvas.

CC BY

Desenvolupament Web

Quan parlem de desenvolupament web fem referència a diferents tecnologies:

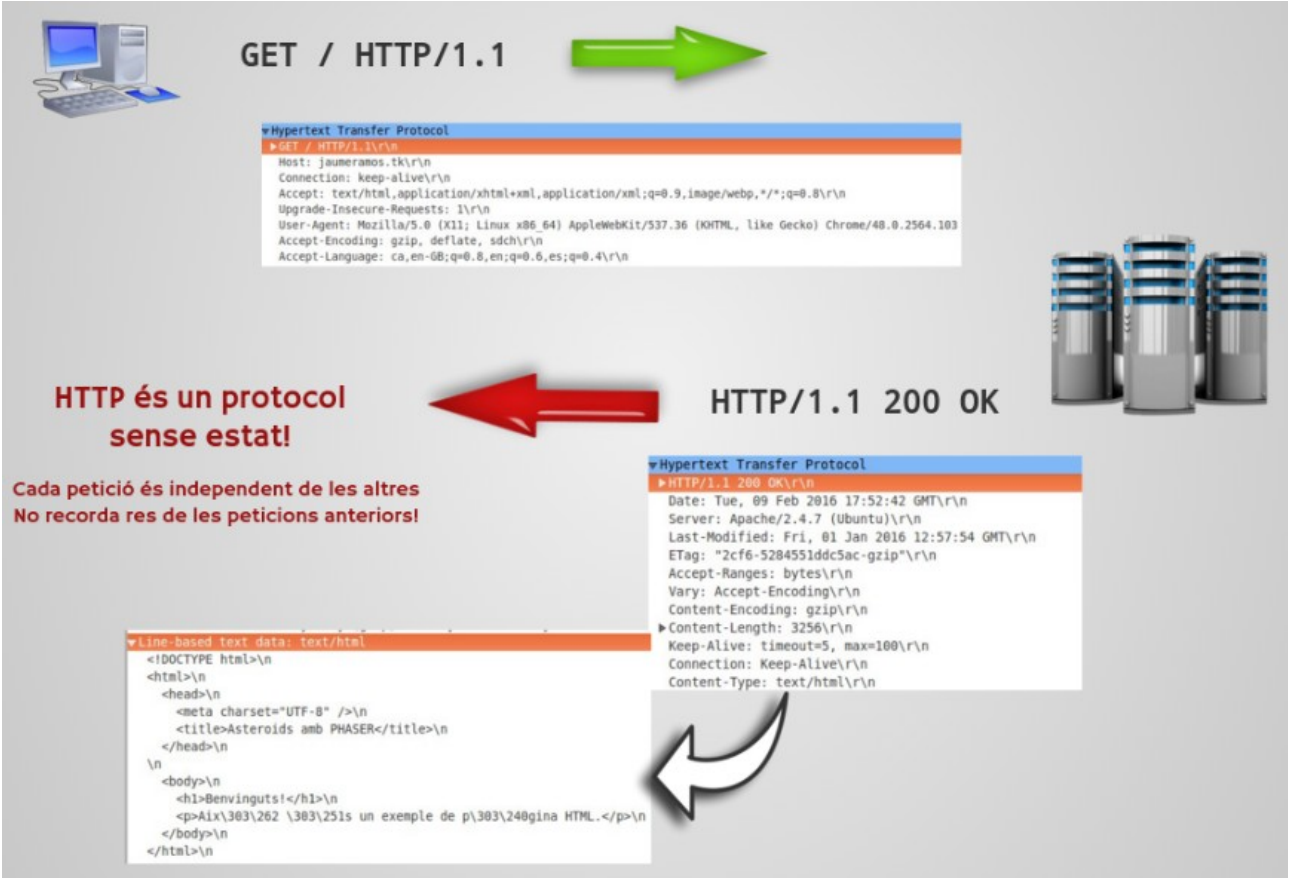
- HTTP: (Hypertext Transfer Protocol) El protocol en què es basa la Web. Fou desenvolupat per Sir Tim Berners-Lee l'any 1986 per permetre la navegació hipertextual per documents formatats amb HTML. La versió utilitzada actualment (HTTP/1.1) fou establerta l'any 1997!
- HTML: (Hyper Text Markup Language) Un llenguatge de marques que permet especificar el contingut d'un document web. Aquest llenguatge ha anat evolucionant molt, fins l'actual versió HTML5 que aporta moltes millores i permet noves funcionalitats a les pàgines web.
- CSS: (Cascading Style Sheets) Permet separar l'estil dels documents del seu contingut, simplificant modificar l'aspecte sense afectar al contingut. La darrera versió CSS3 ha afegit un gran nombre de noves funcionalitats.
- JavaScript: Llenguatge de script desenvolupat per Brendan Eich pel navegador Netscape a l'any 1995

Quan el navegador fa una petició a un servidor web, envia un paquet HTTP demanant un recurs, i el servidor respon amb un altre paquet HTTP que conté el codi HTML del recurs demanat, així com altres dades relacionades com imatges, sons, scripts.... El navegador interpreta aquesta informació i la mostra a l'usuari.

HTTP no fou dissenyat en absolut pensant en les funcionalitats que ara té, i per tant s'han hagut de cercar solucions alternatives per superar algunes de les seves limitacions. Per exemple HTTP és un protocol sense estat, això vol dir que no recorda res de les peticions anteriors, per tant, si volem que l'usuari tingui la percepció que el servidor «recorda» les opcions escollides (usuari, carret de la compra...) li hem de recordar natros. Això es fa intercanviant el navegador i el servidor un fragment d'informació dins el protocol HTTP que anomenem cookie. Aquestes cookies proporcionen al servidor les dades que ell no

CC BY

recorda, i són «perilloses» degut a què desen aquesta informació al nostre ordinador, i si algun atacant maliciós aconsegueix accedir a les cookies podria veure informació important.



En el document HTML hi ha dos grans blocs:

- HEAD: Capçalera del document, hi trobem elements com el joc de caràcters utilitzat (UTF-8, ISO-8859-1...), els fulls d'estil CSS, les llibreries JavaScript...
- BODY: El contingut del document HTML, on els elements poden recolzar-se en fulls CSS per controlar la presentació.

HTML5, la darrera versió d'aquest estàndard, ha aportat moltes millores, entre les que podem destacar:

- Simplifica la sintaxi

CC BY

- Suport per Audio i Vídeo sense necessitar plugins
- Element Canvas per poder dibuixar directament al navegador
- Nous elements que afegeixen info semàntica (article, main, section...)
- Nous elements per entrar dades als formularis (email...)
- Permet a l'usuari editar el contingut d'alguns elements
- Noves funcionalitats per facilitar la validació de les dades entrades
- Local Storage: Permet desar informació sense accedir al sistema de fitxers
- Permet utilitzar GeoLocalització

De la mateixa manera els fulls d'estil CSS també han evolucionat, i la darrera versió, anomenada CSS3, també ha aportat moltes millores:

- Permet seleccionar elements basant-se en el valor dels seus atributs
- Permet arrodonir els cantons de les vores dels elements
- Permet utilitzar una imatge per dibuixar les vores
- Permet afegir ombres i gradients als elements i al text
- Permet afegir transparències (RGB + canal alpha)
- Permet transformar els elements amb rotacions
- Permet afegir Web Fonts a les pàgines

Però potser l'element que més ha influït en l'evolució de la web, que ha passat de servir documents estàtics a oferir aplicacions totalment funcionals dins el navegador, ha estat JavaScript. Aquest llenguatge ha canviat molt des dels seus inicis, i ara que els diferents desenvolupadors de navegadors tenen clar que cal seguir els estàndards, el futur d'aquest llenguatge en el desenvolupament web és molt prometedor.

De forma esquemàtica, algunes de les principals característiques de JavaScript podrien ser:

CC BY

- Creat al 1995 per Brendan Eich a Netscape Corporation
- No té cap relació amb Java (apart del nom)
- Llenguatge de Script (o Interpretat)
- Actualment es troba estandarditzat per la ECMA amb el nom EcmaScript
- Dèbilment tipat (no cal definir el tipus de variables)
- Multiparadigma: Orientació a Objectes, Imperatiu, Funcional
- Basat en prototipus enlloc de classes
- Pràcticament tot en JS són "objectes"
- Els objectes en JS són arrays associatius (parells nom-valor)
- Les funcions són elements first-class (de fet són objectes)
- Suporta funcions anònimes i closures
- La sintaxi és molt flexible, podent fer el mateix de moltes formes diferents!!!
- Els navegadors actuals fan servir compilació just-in-time per millorar la eficiència
- Disposa d'entorns d'execució fora del navegador (node.js)

La darrera versió estandarditzada de JavaScript s'anomena EC6 (EcmaScript V6) encara que també es coneix amb el nom de «Harmony» ja que ha estat fruit de l'acord entre els diferents desenvolupadors de navegadors. No tots els elements definits en aquest estàndard es troben actualment implementats en la majoria de navegadors. Avui en dia la majoria ofereixen una implementació complerta de EC5, afegint progressivament nous elements de EC6.

CC BY

Els Game Loops (Bucles del Joc)

Actualment alguns dels llenguatges més utilitzats per desenvolupar jocs són C++ i C#, aquest darrer especialment per la popularització del framework 3D Unity, basat en aquest llenguatge. La raó de fer servir aquests llenguatges és que permeten un millor control dels processos per tal de millorar el rendiment dels jocs desenvolupats.

Darrerament el llenguatge JavaScript ha estat molt optimitzat, i actualment és una alternativa totalment vàlida per desenvolupar jocs amb uns requeriments gràfics no massa elevats. Fins i tot hi ha frameworks que permeten desenvolupar jocs 3D[3] [4] amb JavaScript.

La flexibilitat que aporta JavaScript per poder desenvolupar jocs multiplataforma que puguin utilitzar-se en qualsevol mena de dispositiu que tingui un navegador web, fa que hi hagi un gran desenvolupament en els frameworks JavaScript per crear jocs.

Independentment de l'entorn de desenvolupament utilitzat, tots els jocs tenen una estructura semblant pel que fa al seu funcionament.

Els jocs són diferents a la majoria d'aplicacions d'ordinador, en aquestes normalment l'aplicació no fa res fins que l'usuari no fa alguna acció, sigui amb el teclat, el ratolí o

3 http://www.babylonjs.com/
4 http://threejs.org/

CC BY

qualsevol altre dispositiu, per tant la CPU està molt temps esperant rebre un event generat per l'usuari per tal de processar-lo.

```
while (true)
{
   Event e = esperarEvent();
   processarEvent(e);
}
```

Els jocs, pel contrari estan canviant de forma contínua, tot i que no hi hagi cap acció per part de l'usuari. Per tal en un joc cal implementar un bucle que vagi actualitzant de forma contínua els elements del joc. Aquesta actualització cal fer-la a un ritme adient, ja que si ho féssim massa ràpid el joc no podria ser jugat, mentre que si ho féssim massa a poc a poc el joc no mostraria un moviment uniforme, trencant la sensació de realitat que busca la immersió del jugador en el joc. Els bucles dels jocs també capturen els events generats pels usuaris, però no esperen que succeeixin, el bucle es repeteix de forma continuada, i cada cop es comprova si hi ha hagut alguna acció per part de l'usuari per tal d'actuar en conseqüència.

Els bucles de jocs han anat evolucionant amb la tecnologia. En un principi, on hi havia equips monousuari i monotasca, amb un únic processador, el joc era l'únic procés que s'executava a la màquina i per tant hi tenia el control total. En aquesta època era habitual trobar bucles amb un codi semblant a aquest:

```
while (true)
{
   processarEntradaUsuari();
   actualitzarElementsJoc();
   mostrarJoc();
}
```

Aquest bucle depèn de la velocitat del processador, i de vegades pot ser massa ràpid, de forma que calia introduir algun element per controlar la velocitat. Una forma de fer-ho pot ser la següent:

CC BY

```
while (true)
{
   processarEntradaUsuari();
   actualitzarElementsJoc();
   MostrarJoc();

   for (i=1;i<1000;i++){}
}
```

Hem afegit un bucle que no fa res més que ocupar uns quants cicles de la CPU per disminuir la velocitat. El problema d'aquest bucle és que la seva velocitat pot variar en funció de la màquina en què l'executem, i per tant l'experiència de l'usuari pot variar en funció d'on es jugui. A més a més en els equips actuals, que poden fer moltes tasques a l'hora, poden ralentitzar el funcionament de les tasques que es fan en segon pla. Una alternativa més eficient és utilitzar un temporitzador. Això té dos avantatges; la primera és que el programa, mentre espera no consumeix cicles de CPU, i la segona és que permet controlar de forma més exacta la velocitat d'actualització. Molts jocs fan servir una freqüència d'actualització d'uns 60 cops per segon, és a dir que cada segon es mostren 60 fotogrames (o frames en anglès). Aquesta velocitat d'actualització en FPS (Frames Per Second) serà la mateixa independentment de la màquina en què estem jugant. En JavaScript això es pot fer amb la funció setInterval.

```
var FPS = 60;
setInterval(function() {

   processarEntradaUsuari();
   actualitzarElementsJoc();
   MostrarJoc();

}, 1000/FPS);
```

Aquest mètode té un problema, però, ja que si no es pot processar prou ràpid tot el que es fa dins el loop, la màquina treballa per fer molta més feina de la que pot mostrar. A més a més el moviment dins el joc es pot veure alterat ja que els frames descartats provoquen salts. Intentar controlar això en JavaScript resultava una tasca bastant complexa. Per sort

CC BY

HTML5 proporciona una nova API anomenada requestAnimatedFrame que intenta resoldre alguns d'aquests problemes. Aquesta API disposa del mètode requestAnimationFrame que demana al navegador cridar un callback abans de tornar a redibuixar la finestra. Aquest mètode té un únic paràmetre que és el temps en que es fa la crida al callback en milisegons. L'actualització normalment es fa a 60 FPS, tot i que s'intenta respectar la velocitat de refresc del navegador. A part de permetre un moviment més fluid, aquesta API també estalvia cicles de CPU, ja que no s'executa quan l'usuari canvia de pestanya o quan el navegador perd el focus.

```
(function rAF(tempsMiliSegons){
   processarEntradaUsuari();
   actualitzarElementsJoc();
   MostrarJoc();

   requestAnimationFrame(rAF);
 })();
```

Si voleu un exemple on veure el funcionament de requestAnimationFrame, enganxeu el següent codi en un document HTML.

```
<!doctype html>
<html>
<head>
   <title>requestAnimationFrame</title>
   <style>
      div {
         width: 10px;
         height: 10px;
         background: orange;
         float: left;
      }
   </style>
</head>
<body>
  <script type='text/javascript'>
    (function rAF(){
       document.body.appendChild(document.createElement('div'));
       requestAnimationFrame(rAF);
     }
    )();
  </script>
</body>
```


CC BY

Per sort Phaser ens soluciona aquest problema, tal i com veurem al següent apartat. Quan hi és disponible fa servir requestAnimationFrame, i si no es pot utilitza la funció setTimeOut per definir el bucle del joc a 60FPS.

Podeu trobar informació addicional sobre els temporitzadors i els bucles en JavaScript en el blog de Isaac Sukin[5].

5 http://www.isaacsukin.com/news/2015/01/detailed-explanation-javascript-game-loops-and-timing

CC BY

Desenvolupament d'un joc amb Phaser

Phaser és una llibreria JavaScript que proporciona un framework complet per desenvolupar jocs. Inclou des dels elements més bàsics com pot ser el bucle del joc i el control de l'element on es mostra el joc, fins a elements molt més avançat com els càlculs de les lleis físiques per controlar el moviment dels elements del joc.

L'objecte Game és l'element central d'un joc amb Phaser. Per crear un joc només necessitem crear un objecte Game i indicar-li els callbacks o funcions que s'encarregaran de gestionar cada element del bucle del joc:

```
var game = new Phaser.Game(
        amplada, altura,      // Mida inicial del joc
        Phaser.CANVAS,        // Forma de mostrar el joc
        'contenidor',         // Element HTML on mostrar-lo
        {
            preload: ferPreload,      //Callback per carregar elements
            create:  ferCreate,       //Callback per la configuració inicial
            update:  ferUpdate        //Callback per actualitzar el joc
        }
);

function ferPreload(){
    // Codi per carregar els elements gràfics i assignar-lis una clau
}

function ferCreate(){
    // Codi per afegir al joc els elements gràfics necessaris
}

function ferUpdate(){
    // Codi que s'executara 60FPS per tal d'actualitzar el joc
}
```

La mida inicial del joc pot ser un valor fixe, o podem agafar la mida actual de la finestra on es mostrarà al navegador. Això ho podem fer amb les propietats del element window o el element document. La mida del joc en un navegador d'escriptori no suposa cap problema, però quan volem tenir en compte la mida en els dispositius mòbils la cosa es pot complicar molt, degut a les enormes diferències de mides i resolució gràfica que hi ha entre ells. De

CC BY

moment, i per fer-ho senzill, utilitzarem una mida fixa.

Els diferents callbacks que podem passar a Phaser quan creem un joc són:

- **init**: Es crida en primer lloc al iniciar el joc.
- **preload**: Serveix principalment per carregar els elements gràfics. Encara no ha començat el Game Loop.
- **loadUpdate**: Durant preload, permetria mostrar una barra de progrés mentre es carrega el joc.
- **loadRender**: Durant preload, no acostuma a utilitzar-se, tot el codi es pot posar en loadUpdate. Permetria separar totalment el codi que actualitza el joc (update) del que el mostra (render).
- **create**: Es crida en acabar preload i abans d'iniciar el Game Loop. Permet crear tots els elements gràfics a partir dels objectes carregats en el preload. Ha de contenir tot el codi per iniciar el joc.
- **update**: Cridada per cada frame (per defecte 60FPS) és on hem de posar la majoria del codi del nostre joc: Capturar l'input de l'usuari, actualitzar la posició dels elements, comprovar-hi si hi ha col·lisions....
- **render**: Cridada després de mostrar d'actualitzar el joc (CANVAS o WEBGL) i permet afegir efectes post-renderitzat. Moltes vegades s'utilitza per sobreposar informació de depuració al joc.
- **resize**: Només es crida en cas de modificar les dimensions del contenidor del joc, sempre que aquest s'hagi creat amb el mode RESIZE. Rep dos paràmetres: Amplada i altura del nou contenidor. Permet reposicionar els elements del joc per fer jocs responsive.
- **shutdown**: Es crida al tancar un estat del joc, bé per acabar el joc, bé per passar a un altre estat.

Amb HTML5 podem mostrar gràfics en l'element Canvas, però hi ha navegadors que suporten la API WebGL que permet utilitzar gràfics 3D dins el canvas i que aprofita la potència gràfica de la GPU. Si el navegador ho suporta, WebGL acostuma a ser més

CC BY

eficient que treballar directament amb el canvas. Al crear un joc amb Phaser podem indicar-li quin mode de renderitzat volem amb tres constants predefinides:

```
Phaser.CANVAS     // Dibuixa dirèctament al Canvas

Phaser.WEBGL      // Fa servir la API WEBGL

Phaser.AUTO       // Escull la millor alternativa
```

Al crear el joc, Phaser injecta un element Canvas, però podem indicar-li dins quin element HTML volem fer-ho. Això permet configurar aquest element amb estils CSS. Si no s'indica cap element HTML Phaser crea el canvas directament en l'element Body.

La següent pàgina HTML que crea un joc Phaser genera un canvas en el body.

```
<!doctype html>
<html>
  <head>
    <script src="js/phaser.min.js" type="text/javascript"></script>
    <script type="text/javascript">

      var game = new Phaser.Game(400, 500, Phaser.AUTO, '',
        {preload: ferPreload, create: ferCreate, update: ferUpdate});

      function ferPreload(){}
      function ferCreate(){}
      function ferUpdate(){}

    </script>
  </head>
  <body>
  </body>
</html>
```

Si mirem el codi HTML generat per Phaser veurem això:

```
<body>
<canvas width="400" height="500" style="display: block; touch-action:
none; -webkit-user-select: none; -webkit-tap-highlight-color: rgba(0,
0, 0, 0); width: 400px; height: 500px; cursor: inherit;">
</canvas>
</body>
```


CC BY

Asteroids amb Phaser i JavaScript

Per desenvolupar una versió del joc arcade Asteroids, seguirem una sèrie de passos de forma que cada cop anirem afegint nous elements i funcionalitats al joc.

Cada pas el podeu trobar en un fitxer step_n.html de manera que podeu anar seguint pas a pas els canvis fets. El resultat final el teniu en el document asteroids.html. Us recomano que intenteu fer vosaltres els passos un darrera l'altre, no hi ha cap altra forma per aprendre a programar que escriure el codi per vosaltres mateixos.

Passos per desenvolupar Asteroids:

1. Carregar el fons i animar-lo.
2. Dibuixar la nau i associar-li el motor de la física ARCADE.
3. Controlar el moviment de la nau amb els cursors.
4. Controlar la sortida de la nau per la pantalla, de forma que torni a entrar per la part oposada.
5. Fer que la nau dispari al prémer la tecla espai. Hi ha d'haver un nombre limitat de bales, i han de tenir un temps de vida fins que desapareixien, per evitar omplir la pantalla de bales. Cal deixar un temps entre bala i bala.
6. Fer aparèixer un asteroide cada segon, controlant que no surti massa prop de la nau. Cal revisar que si l'asteroide surt de la pantalla torni a aparèixer per l'altra banda.
7. Determinar si hi ha col·lisions bala-asteroide, i si n'hi ha mostrar una explosió i actualitzar la puntuació del joc.
8. Determinar si hi ha col·lisions nau-asteroide, i si n'hi ha mostrar una explosió i restar una vida.
9. Mostrar la puntuació del joc i el nombre de vides restant.
10. Aturar el joc quan no quedin més vides.

CC BY

Step_1: Carregar el fons i animar-lo

```
// Variables pel fons del joc
var fonsNebulosa, fonsPedres;

// Crea el joc i associa les funcions de Callback
var game = new Phaser.Game(600, 800, Phaser.AUTO, '',
    {preload: ferPreload, create: ferCreate, update: ferUpdate});

function ferPreload(){

    // Carrega els elements gràfics pel fons
    game.load.image('nebulosa', 'assets/nebulosa.png');
    game.load.image('pedres', 'assets/pedres.png');

}

function ferCreate(){

    //  Posa el fons del joc
    game.stage.backgroundColor = '#000000';
    fonsNebulosa = game.add.tileSprite(0, 0,
                               game.width, game.height, 'nebulosa');
    fonsPedres = game.add.tileSprite(0, 0,
                               game.width, game.height, 'pedres');

}

function ferUpdate(){

    //  Mou cap cap a la dreta les pedres
    fonsPedres.tilePosition.x += 0.5;

}
```


CC BY

Step_1 - Explicació

Hem de definir dos variables globals per desar la imatge del fons. Hi ha dos elements, un fons amb una imatge d'una nebulosa, i un altra imatge amb unes pedres de diferents mides que anirem desplaçant per la pantalla per crear la il·lusió d'un fons que es mou.

El primer que hem de fer és crear el joc. En aquest cas hem posat una mida fixa de 600x 800 pixels, però si haguessim volgut ajustar la mida del joc a l'espai disponible en el navegador, hauriem pogut crear el joc a partir de la informació de les propietats window.innerWidth i window.innerHeight. També podríem indicar en quin element HTML volem que Phaser dibuixi el joc. Si no especifiquem res ho posa dins el <body>.

```
var game = new Phaser.Game(
    window.innerWidth,
    window.innerHeight,
    Phaser.AUTO,
    'ElementHTML',
    {preload: ferPreload,
     create:  ferCreate,
     update:  ferUpdate
    }
);
```

Un cop creat el joc hem de programar els callbacks. A la funció **ferPreload** hem de posar el codi per carregar els elements del joc: sprites, fons, música, tipus de lletra....

De moment carregarem les dos imatges del fons. Ho fem amb la funcio **load.image**, i li hem de passar com a paràmetres la ubicació de l'arxiu d'imatge i el nom amb que farem referència al nostre codi a aquest element.

Un cop carregats, a la funció ferCreate hem de dibuixar-los. Primer definim el fons del joc (**game.stage.backgroundColor**) amb color negre (**#000000**) i després afegim les imatges de la nebulosa i les pedres amb la funció **game.add.tileSprite** que requereix els següents paràmetres: x, y, ample, alt i nom de l'objecte (que abans haurem carregat). Les posicions x i y són relatives al contenidor on es mostra la imatge, que pot ser el mateix joc o un grup

CC BY

d'imatges que es mouen de forma coordinada. Les propietats ample i alt fan referència a la mida amb que es vol mostrar la imatge, no la seva mida real. Phaser ajusta la mida mostrant una part o repetint la imatge.

Per acabar a la funció **ferUpdate** desplaçarem el fons horitzontalment incrementant la posició x de la imatge mitjançant la propietat **tilePosition.x.**

CC BY

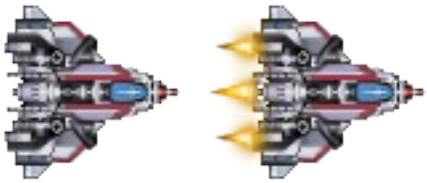
Step_2: Dibuixar la nau i associar-li la física ARCADE

```
// Variable per la nau
var nau;

function ferPreload(){

      // Carrega la imatge de la nau. Hi ha dos imatges:
      //una quan acelera i l'altra amb els motors apagats
      game.load.spritesheet('nau', 'assets/nau-x2.png', 90, 90);
}

function ferCreate(){

      // Dibuixa la nau
      nau = game.add.sprite(game.width / 2, game.height / 2, 'nau');

      // Indica el punt de referència de la nau
      nau.anchor.x = 0.5;
      nau.anchor.y = 0.5;

      // Inicialitza el motor de física ARCADE
      game.physics.startSystem(Phaser.Physics.ARCADE);

      // Associa el motor físic amb la nau per controlar-la
      game.physics.enable(nau, Phaser.Physics.ARCADE);

      // Defineix el fregament de la nau
      nau.body.drag.set(100, 100);
      nau.body.angularDrag = 200;

      // Defineix la màxima velocitat de la nau
      nau.body.maxVelocity.set(200, 200);
      nau.body.maxAngular = 200;

}
```


CC BY

Step_2 - Explicació

La nau és una imatge especial, ja que està formada per dos imatges juntes, una amb els motors «engegats» i un altra amb els motors «aturats». En els jocs és una tècnica molt utilitzada per simular moviment. Una imatge està formada per una seqüència de frames que poden donar lloc a una animació. Phaser carrega aquestes imatges amb la funció **load.spriteSheet**, semblant a **load.image**, però requerint a més a més informar de l'amplada i l'alçada dels frames de la imatge.



Un cop carregada la imatge de la nau, hem de dibuixar-la, aquest cop farem servir la funció **add.Sprite**, semblant a **add.tileSprite**. Un Sprite no es repeteix com un tileSprite fins omplir la mida demanada, de fet per afegir un Sprite només cal indicar la posició dins el joc (x, y) i la clau de l'element gràfic a mostrar.

Per simplificar el posicionament dels sprites en el joc, aquestos disposen d'una propietat anomenada anchor, que fa referència al punt de la imatge que es situarà en la coordenada indicada pel sprite.

(0, 0) → Cantó superior esquerre

(0.5, 0.5) → Centre de la imatge

(1, 1) → Cantó inferior dret

A phaser els punts s'indiquen amb les coordenades x i y, i es poden indicar de diverses formes; fent servir les propietats x i y o fent servir els mètodes set i setTo (fan exactament el mateix).

CC BY

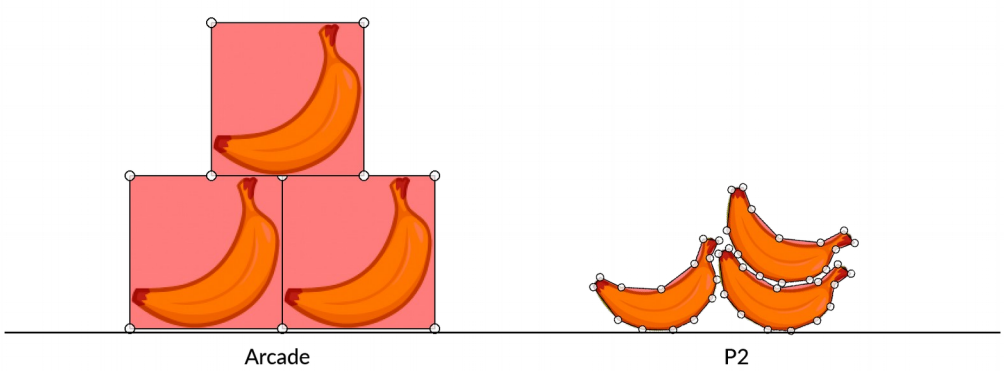
```
nau.anchor.x = 0.5;
nau.anchor.y = 0.5;

nau.anchor.set(0.5);
nau.anchor.set(0.5, 0.5);

nau.anchor.setTo(0.5);
nau.anchor.setTo(0.5, 0.5);
```

Un cop creada la nau i afegida al nostre joc, hem de definir-li el motor de física que volem fer servir. Per això primer hem d'inicialitzar el motor de física que necessitem amb la funció **game.physics.startSystem** i indicant-li com a paràmetre el motor desitjat:

Phaser.Physics.ARCADE: Motor senzill i molt ràpid. Detecta les col·lisions comparant un quadre que agafa tot el sprite (si tenen formes irregulars els resultats no són perfectes). Aquestes col·lisions s'anomenen «AABB» i no tenen en compte si dins el quadre hi ha imatge o és un àrea transparent.

Phaser.Physics.NINJA: Més avançada que ARCADE (i més lenta) gestiona les rotacions i permet treballar amb pendents i formes més complexes.

Phaser.Physics.P2: Més sofisticat que ARCADE i NINJA, i per tant requereix de més recursos. Permet associar a un sprite un conjunt de punts que determinen la seva forma, d'aquesta forma la detecció de col·lisions és més precisa. Aquesta forma de detectar les col·lisions s'anomena «full body». També modela molts més elements, permetent crear jocs més complexos.



CC BY

Un cop inicialitzat el motor de física, l'hem d'associar als sprites on ens calgui controlar els seus moviments. Hi ha elements que no necessitaran tenir associat cap motor, com per exemple els elements estàtics, els que tinguin moviments aleatoris, etc. Per associar un motor de física a un sprite hem de fer servir la funció **physics.enable** i passar-li com a paràmetre l'objecte del sprite i el motor que volem utilitzar, és possible fer servir més d'un motor en un mateix joc. Els diferents motors funcionen creant un «body» sobre el sprite de forma que els càlculs es fan sobre aquest element, i les propietats físiques també s'assignen a aquest element.

Al nostre joc, afegirem valors per les propietats que controlen el fregament (**drag** i **angularDrag**) i la velocitat màxima del sprite (**maxVelocity** i **maxAngular**). Amb això farem que la nau s'aturi de forma progressiva, quan no fem servir els controls de moviment, i evitarem que la velocitat sigui massa alta per fer el joc injugable. Les propietats **drag** i **maxVelocity** necessiten els valors (x, y) del vector corresponent. De fet es representen amb un objecte Phaser.Point.

CC BY

Step_3: Controlar el moviment de la nau amb els cursors

```
// Variables per les tecles que controlen el joc
var acelerar, girarDreta, girarEsquerra;

function ferCreate() {

   /* Step_3 */
   // Controla les tecles que gestionen el moviment de la nau
   acelerar = game.input.keyboard.addKey(Phaser.Keyboard.UP);
   acelerar.onDown.add(function() {
       game.physics.arcade.accelerationFromRotation(
            nau.rotation, 200, nau.body.acceleration
       );
       nau.frame = 1;
   });
   acelerar.onHoldCallback = (function() {
       game.physics.arcade.accelerationFromRotation(
            nau.rotation, 200, nau.body.acceleration
       );
   });
   acelerar.onUp.add(function() {
       nau.body.acceleration.set(0, 0);
       nau.frame = 0;
   });
   girarDreta = game.input.keyboard.addKey(Phaser.Keyboard.RIGHT);
   girarDreta.onDown.add(function() {
       nau.body.angularAcceleration = 100;
   });
   girarDreta.onUp.add(function() {
       nau.body.angularAcceleration = 0;
   });

   girarEsquerra = game.input.keyboard.addKey(Phaser.Keyboard.LEFT);
   girarEsquerra.onDown.add(function() {
       nau.body.angularAcceleration = -100;
   });
   girarEsquerra.onUp.add(function() {
       nau.body.angularAcceleration = 0;
   });

   // Evita que les tecles es propaguin al navegador
   this.game.input.keyboard.addKeyCapture([Phaser.Keyboard.UP,
        Phaser.Keyboard.RIGHT, Phaser.Keyboard.LEFT
   ]);

}
```


CC BY

Step_3 - Explicació

Aquesta versió de Asteroids es controla amb el teclat, i per tant hem de «capturar» les tecles que ens interessen per tal de poder respondre als events generats per l'usuari, i associar un callback a aquests events.

Això ho hem de fer amb dos funcions Phaser.

Per capturar els events produïts en una tecla necessitem la funció **game.input.keyboard.addKey**, a la que li hem de passar com a paràmetre la tecla que ens interessa. Per això es poden fer servir les constants definides per Phaser. Les que ens interessen pel nostre joc són:

Phaser.Keyboard.UP

Phaser.Keyboard.RIGHT

Phaser.Keyboard.LEFT

Aquesta funció ens retorna un objecte amb el que podrem comprovar l'estat de la tecla associada i també assignar-li callbacks per respondre a diferents events. Els events que ens interessen són:

- **onDown:** Es dispara quan es prem una tecla.
- **onUp**: Es dispara quan es deixa de prémer una tecla.

A aquests events els associarem un callback que serà una funció anònima que gestionarà el que cal fer en cada cas:

Al prémer les tecles dreta o esquerra cal modificar l'acceleració angular per que la nau giri, mentre que al deixar de prémer aquestes tecles hem de posar l'acceleració angular a zero. Recordem que havíem definit fregament angular, i això farà que la nau deixi de girar de forma progressiva.

De la mateixa forma al prémer la tecla amunt cal donar valor a l'acceleració i al deixar de prémer-la posarem a zero aquesta acceleració. El fregament definit farà que la nau s'aturi progressivament.

CC BY

Com l'event **onDown** només es dispara al premer la tecla, l'acceleració definida té la direcció de la nau en el moment de prémer la tecla. Si volem que en cada moment l'acceleració tingui la mateixa direcció que la nau, hem de passar un callback a la propietat **onHoldCallback**, que farà que s'actualitzi l'acceleració en la direcció de la nau en cada moment.

I no hem de fer res més!!!!! El framework Phaser, amb el seu motor de física ARCADE s'encarregarà de calcular en cada moment la velocitat i posició de la nau, a partir d'aquests paràmetres.

Si volem evitar que les tecles premudes es propaguin al navegador i facin la funció definida, podem aturar el processament dels events (bubbling) amb la funció **`game.input.keyboard.addKeyCapture`**.

CC BY

Step_3a: Controlar el moviment de la nau amb els cursors

De forma alternativa podem controlar el moviment en el ferUpdate amb les variables on hem capturat les tecles. També podem capturar de forma genèrica totes les tecles dels cursors en una única variable

```
// Variables per les tecles que controlen el joc
var cursors;

function ferCreate() {

    // Controla les tecles que gestionen el moviment de la nau
    cursors = game.input.keyboard.createCursorKeys();

    // Evita que les tecles es propaguin al navegador
    this.game.input.keyboard.addKeyCapture([Phaser.Keyboard.UP,
                                            Phaser.Keyboard.RIGHT,
                                            Phaser.Keyboard.LEFT]);

}

function ferUpdate() {

    if (nau.alive) {
        //  Tecla amunt fa accelerar la nau
        if (cursors.up.isDown) {
            game.physics.arcade.accelerationFromRotation(nau.rotation,
                                        200, nau.body.acceleration);
            nau.frame = 1;
        } else {
            nau.body.acceleration.set(0, 0);
            nau.frame = 0;
        }
        // Tecles dreta i esquerra fan rotar la nau
        var accelAng = 0;  // Si es premem les dos a l'hora no fa res
        if (cursors.left.isDown) {
            accelAng = -100;
        }
        if (cursors.right.isDown) {
            accelAng += 100;
        }
        nau.body.angularAcceleration = accelAng;
    }

}
```


CC BY

Step_3a - Explicació

Definir un callback sobre els events **onDown** fa que el codi només s'executi un cop al prémer la tecla. Això fa que si es manté la tecla UP premuda, l'acceleració es continuï aplicant en la mateixa direcció en que la nau estava al començar a prémer la tecla.

Una alternativa seria passar-li un callback a la propietat **onHoldCallback** de manera que si la tecla es queda premuda, cada cop l'acceleració tingui la direcció actual de la nau.

Un altra forma de controlar les tecles és capturar-les només en la funció **ferCreate** i després preguntar per la situació de cada tecla dins la funció **ferUpdate**. Aquesta forma permet un control més acurat, ja que la comprovació es fa un cop per cada frame, i a més a més permet controlar combinacions múltiples de tecles (dos tecles premudes a l'hora, etc). També pot ser més lent ja que repeteix el codi de control en cada frame, mentre que si en tenim prou amb associar el codi als events **onUp** i **onDown**, aquest codi només s'executa quan es prem o es deixa de prémer la tecla.

CC BY

Step_4: Controlar la sortida de la nau per la pantalla

Es tracta que la nau torni a entrar a la pantalla per la part oposada per on ha sortit.

```
function ferUpdate() {

    // Verifica si la nau surt de la pantalla
    surtPantalla(nau);

}


function surtPantalla(sprite) {

    // Comprova si surt horitzontalment
    if (sprite.x < 0) {
        sprite.x = game.width;
    } else if (sprite.x > game.width) {
        sprite.x = 0;
    }

    // Comprova si surt verticalment
    if (sprite.y < 0) {
        sprite.y = game.height;
    } else if (sprite.y > game.height) {
        sprite.y = 0;
    }
}
```

Aquest codi el posarem en una funció, ja que ens caldrà reutilitzar-lo per fer que els asteroides facin el mateix que la nau. Per tant definirem una funció que rebi com a paràmetre un sprite qualsevol i de moment la cridarem passant-li la nau.

El que hem de fer a la funció és comprovar si la nau surt de la pantalla. Realment el que verifiquem és si la nau surt de les dimensions definides pel joc (**game.width** i **game.height**) en el moment de crear-lo.

CC BY

El joc es desenvolupa en un canvas que va des de la posició (0, 0) corresponent al cantó superior esquerre, a la posició (game.width, game.height) corresponent al cantó inferior dret. Hem de verificar tant per l'eix horitzontal (x) com pel vertical (y) que quan la posició surti d'aquests límits el sprite vagi a l'altre extrem.

CC BY

Step_5: Fer que la nau dispari al prémer la tecla espai

Hi ha d'haver un nombre limitat de bales, i han de tenir un temps de vida fins que desapareixien, per evitar omplir la pantalla de bales. Cal deixar un temps entre bala i bala.

Molts teclats tenen problemes per gestionar més de tres tecles premudes alhora. Això s'anomena ghosting, i pot fer que els jocs no es comportin de la forma esperada.

https://www.microsoft.com/appliedsciences/antighostingexplained.mspx

```
function ferPreload() {

    // Carrega la imatge de una bala
    game.load.image('bala', 'assets/bala.png');

}

function ferCreate() {

    //  Crea conjunt de bales amb 30 bales
    bales = game.add.group();
    bales.enableBody = true;
    bales.physicsBodyType = Phaser.Physics.ARCADE;
    bales.createMultiple(30, 'bala');
    bales.setAll('anchor.x', 0.5);
    bales.setAll('anchor.y', 1);

    // Controla la tecla per disparar
    disparar = game.input.keyboard.addKey(Phaser.Keyboard.SPACEBAR);
    disparar.onHoldCallback = function (e) {
        if (game.time.now > seguentBala) {
            bala = bales.getFirstExists(false); // Reviu una bala
            if (bala) {
                bala.reset(nau.body.x + (nau.body.width / 2),
                                nau.body.y + (nau.body.height / 2));
                bala.lifespan = 1000; // Temps de vida (1seg)
                bala.rotation = nau.rotation;
                game.physics.arcade.velocityFromRotation(
                            nau.rotation, 400, bala.body.velocity);
                seguentBala = game.time.now + 80; // Espera 80ms
            }
        }
    };

    // Evita que les tecles es propaguin al navegador
    game.input.keyboard.addKeyCapture([Phaser.Keyboard.SPACEBAR]);

}
```


CC BY

Step_5 - Explicació

Per disparar necessitem els elements gràfics corresponents a les bales, i necessitem controlar els events de la tecla espai, utilitzada per disparar.

La imatge de la bala, com hem fet anteriorment, la carregarem a la funció preLoad, però no l'haurem de fer servir fins al moment que l'usuari premi la tecla de disparar.

Una primera aproximació ens podria fer crear una bala cada cop que disparem, però això ens acabaria omplint tota la pantalla de bales. Cal doncs que les bales tinguin una vida limitada, i que «desapareguin» al cap d'un temps. També semblaria prudent tenir un nombre limitat de bales. Per evitar que les bales surtin massa juntes, i es mostri una línia contínua amb totes les bales arrenglerades, afegirem una demora entre bala i bala.

El que farem doncs, és crear un grup de bales. Un grup és la forma que té Phaser de controlar un conjunt d'elements. Crearem un grup amb 30 bales i les anirem reutilitzant a mida que ens facin falta. Això serà molt més eficient que crear i destruir bales cada cop. El que farem serà amagar (**kill**) i mostrar (**reset**) les bales del nostre grup. Phaser permet definir un temps de vida (**lifespan**) per un sprite, de forma que assignarem un temps de vida de 1000 milisegons (1s) a les nostres bales. Un cop acabat aquest temps Phaser fa desaparèixer la bala i la podrem reutilitzar en un altre «tret».

Les bales també necessiten portar associat el motor de física ARCADE, ja que ens interessarà que Phaser gestioni les col·lisons amb els asteroides per tal de fer-los explotar. En aquest cas, enlloc d'associar el motor físic a cada sprite, el que fem es associar-lo al grup, de forma que cada sprite del grup tindrà el seu «body» de física per controlar el moviment i les col·lisions.

Per crear el grup fem servir la funció **game.add.group** i per crear els elements del grup cridem al mètode **createMultiple** del grup, indicant-li el nombre d'elements a crear, i la clau del sprite a utilitzar.

Si necessitem assignar alguna propietat a tots els elements d'un grup, podem fer servir el mètode **setAll**, indicant-li la propietat a modificar i el seu valor. D'aquesta manera posarem la propietat **anchor** de tots els elements a (0,5, 1) per indicar que la referència de la bala

CC BY

serà el punt al centre inferior de la imatge de la bala.

Un cop creat el grup de bales, hem de capturar la tecla espai per tal de fer que dispari quan es premi. Això ho hem de fer com abans, amb la funció **game.input.keyboard.addKey** i passant-li la constant corresponent a la tecla espai: **Phaser.Keyboard.SPACEBAR** i associant-li un callback per respondre a l'event **onHoldCallback**, no podem fer servir onDown degut a que només s'activaria al prémer la tecla, i no seguiria disparant si la mantenim premuda.

En el codi del callback el que hem de fer és verificar si ja ha passat prou temps des de l'anterior bala, i per això ens ajudarem de la variable seguentBala, on desarem el temps en el que podem disparar una bala (el temps de l'actual bala més una demora que posarem de 80ms). Si ja ha passat aquest temps podem «disparar» una nova bala. La comprovació del temps actual la fem amb la funció **game.time.now**.

Per disparar una bala hem d'agafar la primera bala disponible dins el grup (és a dir que no sigui visible) el mètode **getFirstExists**(false) ens torna la primera bala inactiva (false). Si no n'hi ha cap no farem res, i si trobem una bala inactiva la reviurem cridant al seu mètode reset i indicant-li la posició, que haurà de correspondre al centre de la nau. Un cop activa la bala, li definim el seu temps de vida (**lifespan**) i li donem la mateixa direcció que la nau (**nau.rotation**) i li donem com a velocitat del tret 400px/s. Per tal que la velocitat de la bala sigui l'esperada hem de recolzar-nos en la funció del motor de física **game.physics.arcade.velocityFromRotation** indicant-li com a paràmetres la rotació de la nau i la velocitat lineal que volem. Aquesta funció defineix la velocitat en els eixos x i y i ens estalvia fer els càlculs trigonomètrics necessaris.

Per acabar definim quan podrà ser disparada la següent bala, posant a la variable seguentBala un valor 80ms superior al temps actual.

I ja ho tenim! Només ens queda aturar el processament de la tecla amb la funció `game.input.keyboard.addKeyCapture`.

Amb Phaser hi ha un altra forma de gestionar l'input amb el teclat. Enlloc de definir callbacks als events, podem simplement capturar la pulsació de les tecles i després a **ferUpdate** preguntar per la situació d'aquestes tecles (**down** o **up**) i actuar en conseqüència. Podeu escollir el mètode que us sigui més còmode.

CC BY

```
var cursors;

function ferCreate(){

    cursors = game.input.keyboard.createCursorKeys();

}

function ferUpdate(){

    if (cursors.up.isDown) {
        /* Accelerar */
    } else {
        /* Aturar acceleració */
    }

}
```


CC BY

Step_6: Fer aparèixer un asteroide cada segon

Cal fer aparèixer un asteroide cada segon, controlant que no surti massa prop de la nau. Cal revisar que si l'asteroide surt de la pantalla torni a aparèixer per l'altra banda.

```
// Variable pel conjunt d'asteroides
var asteroides;
// Variable per demorar un temps fins el següent asteroide
var seguentAsteroide = 0;

function ferPreload() {
    // Carrega la imatge de un asteroide
    game.load.image('asteroide', 'assets/asteroide.png');
}

function ferCreate() {
    // Crea conjunt d'asteroides amb 12 asteroides
    asteroides = game.add.group();
    asteroides.enableBody = true;
    asteroides.physicsBodyType = Phaser.Physics.ARCADE;
    asteroides.createMultiple(12, 'asteroide');
    asteroides.setAll('anchor.x', 0.5);
    asteroides.setAll('anchor.y', 0.5);
}

function ferUpdate() {
    // Comprova el temporitzador per crear un nou asteroide
    if (game.time.now > seguentAsteroide) {
        creaAsteroide();
    }
    // Verifica que no surtin de la pantalla
    asteroides.forEachExists(surtPantalla, this);
}

function creaAsteroide() {
    var asteroide = asteroides.getFirstExists(false);
    if (asteroide) {
        //  Evita que l'asteroide surti massa prop de la nau
        do {
            asteroide.reset(game.rnd.integerInRange(0, game.width),
                            game.rnd.integerInRange(0, game.height));
        } while (game.physics.arcade.distanceBetween(asteroide, nau) < 100);

        asteroide.body.angularVelocity = game.rnd.integerInRange(-300, 300);
        asteroide.body.velocity.setTo(game.rnd.integerInRange(-30, 60),
                                      game.rnd.integerInRange(-30, 60));
        seguentAsteroide = game.time.now + 1000;
    }
}
```


CC BY

Step_6 - Explicació

Els asteroides requereixen un processament semblant a les bales. Haurem de crear un grup d'asteroides i associar-los el motor de física ARCADE.

Per crear un asteroide cada segon, necessitarem una variable seguentAsteroide que, tal com hem fet amb les bales, ens permeti controlar en **ferUpdate** si cal activar un nou asteroide.

La funció **creaAsteroide** l'hem definit fora del **ferUpdate** ja que d'aquesta forma la podríem reutilitzar si volguéssim que a l'iniciar el joc ja hi hagués uns quants asteroides.

Els asteroides els hem de crear en una posició aleatòria dins el canvas, i per tant hem de fer servir la funció **game.rnd.integerInRange** indicant-li com a paràmetres els valors mínims i màxims corresponents a l'amplada i l'alçada del joc. A més a més voldríem evitar que l'asteroide sortís massa prop de la nau, ja que no ens donaria temps de reaccionar, per això farem un bucle calculant posicions aleatòries dins del joc fins que en surti una que estigui allunyada més de 100 pixels de la posició de la nau. El bucle el fem amb l'estructura **do{} while (condició)** i la condició per repetir el bucle serà que la distància nau - asteroide, calculada amb la funció **game.physics.arcade.distanceBetween**, sigui menor que 100 pixels.

Per acabar assignarem a la variable **seguentAsteroide** el valor corresponent al temps actual més 1 segon (1000ms), de forma que el següent asteroide no aparegui fins al cap d'un segon.

CC BY

Step_7: col·lisions bala-asteroide

Determinar si hi ha col·lisions bala-asteroide, i si n'hi ha mostrar una explosió i actualitzar la puntuació del joc.

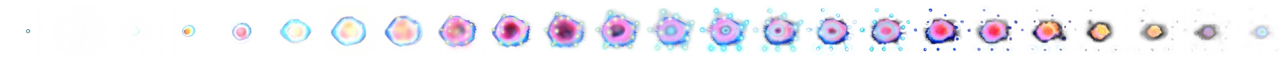
```
// Variable pel conjunt d'explosions
var explosions;

// Variable per comptar els punts de la partida
var punts = 0;

function ferPreload() {
    // Carrega la imatge per fer una explosió
    game.load.spritesheet('explosio', 'assets/explosio.png', 128, 128);
}

function ferCreate() {

    // Crea conjunt d'explosions
    explosions = game.add.group();
    explosions.createMultiple(12, 'explosio');
    explosions.setAll('anchor.x', 0.5);
    explosions.setAll('anchor.y', 0.5);
    explosions.forEach(function (explosio) {
        explosio.animations.add('explosio');
    }, this);

}

function ferUpdate() {

    //  Revisa les col·lisions entre el bales i asteroides
    game.physics.arcade.overlap(bales, asteroides,function (bala, asteroide) {
        //  Amaga la bala i l'asteroide
        bala.kill();
        asteroide.kill();
        //  Crea una explosió
        var explosio = explosions.getFirstExists(false);
        explosio.reset(asteroide.body.x, asteroide.body.y);
        explosio.play('explosio', 30, false, true);
        //  Suma punts
        punts += 10;
    }, null, this);

}
```


CC BY

Step_7 - Explicació

Per mostrar una animació Phaser necessita carregar tots els frames que la formen. Això ho fem, com en el cas de la nau, amb la funció **game.load.spritesheet** indicant-li l'element gràfic amb els frames de l'animació i la mida de cada frame.

Aquí podem veure un exemple d'animació que representa una explosió. Per tal que es mostri de forma adient cal especificar o la mida de cada fotograma (frame) o bé el nombre total de fotogrames. Al reproduir una animació cal especificar el nombre de FPS que cal mostrar, de forma que Phaser escollirà el frame adient en funció del temps. Una animació es pot reproduir només un cop o ser repetida sense parar en un bucle.



Per les explosions farem servir un grup, igual que amb els asteroides i les bales. Com no és probable que els 30 asteroides siguin destruïts alhora, potser amb només 12 explosions ja en tindrem prou.

Per associar una animació a un sprite hem de fer servir el mètode **animations.add** i indicar-li el nom de l'element gràfic corresponent a l'animació. D'aquesta forma podrem reproduir aquesta animació cridant al mètode **play** del sprite. Com que volem fer el mateix per tots els elements del grup d'explosions, podem fer servir el mètode forEach que permet executar un callback sobre cada element del grup.

```
explosions.forEach(function (explosio) {
    explosio.animations.add('explosio');
}, this);
```

Per revisar les col·lisions hem de fer servir la funció **game.physics.arcade.overlap** a la que li podem passar dos grups d'objectes i un callback que serà cridat en cas que es detecti una col·lisió entre algun dels objectes dels grups.

En cas de detectar una col·lisió entre una bala i un asteroide, el que hem de fer és amagar

CC BY

els dos elements gràfics, cridant al mètode **kill**, agafar la primera explosió disponible, reviure-la en la posició de l'asteroide que ha col·lisionat amb la bala, i reproduir l'animació que representa l'explosió cridant al mètode **play** indicant-li el nom de l'animació. Aquest mètode requereix tres paràmetres: El frameRate a que es reprodueix l'animació(30 fps), si l'animació ha de repetir-se en un bucle (false), i si cal amagar el sprite pare de l'animació un cop acabi (true).

Per acabar sumarem 10 punts per cada asteroide «destruït».

CC BY

Step_8: col·lisions nau-asteroide

Determinar si hi ha col·lisions nau-asteroide, i si n'hi ha mostrar una explosió i restar una vida.

```
// Variable per comptar les vides que ens queden
var vides = 3;

function ferUpdate() {

    //  Revisa les colisions entre nau i asteroides
    game.physics.arcade.overlap(nau, asteroides, function (nau, asteroide) {
        // Fa desapareixer l'asteroide
        asteroide.kill();
        // Mostra una explosicó
        var explosio = explosions.getFirstExists(false);
        explosio.reset(asteroide.body.x, asteroide.body.y);
        explosio.play('explosio', 30, false, true);
        vides--;
    }, null, this);

}
```

El que hem de fer és exactament el mateix que en el pas anterior. Si es detecta una col·lisió entre la nau i un asteroide, cal mostrar una col·lisió i actualitzar el comptador de vides. El joc segueix a la nau al mateix lloc, per això no cal fer res amb la nau.

CC BY

Step_9: Mostrar la puntuació del joc i el nombre de vides

```
var textPunts, textVides;

function ferCreate() {

    //  Afegim la informació dels punts i les vides
    textPunts = game.add.text(10, 10, "Punts: " + punts, {
        font: '34px Arial',
        fill: '#fff'
    });
    textVides = game.add.text(game.width-150, 10, "Vides: "+vides, {
        font: '34px Arial',
        fill: '#fff'
    });

}


function ferUpdate() {

    //  Afegim la informació dels punts i les vides
    textPunts.text = "Punts: " + punts;
    textVides.text = "Vides: " + vides;

}
```

Per mostrar un text amb Phaser primer hem de crear un objecte de text amb la funció **game.add.text** indicant-li com a paràmetres les coordenades (x, y) on ha de mostrar-se el text, el contingut del text a mostrar, i un objecte amb les propietats del text, bàsicament el tipus de lletra (**font**) i el color del text (**fill**). Com la posició i el tipus de lletra no canviaran en tot el joc, això ho podem fer només un cop a la funció **ferCreate**.

Un cop creat l'objecte podem modificar el text a mostrar amb la propietat text, de forma que a ferUpdate podem actualitzar la informació amb el temps i les vides.

Amb JavaScript podem juntar textos amb l'operador «**+**».

CC BY

Step_10: Aturar el joc quan no quedin més vides

```
function ferUpdate() {

    game.physics.arcade.overlap(nau, asteroides, function (nau, asteroide) {

        /* .....  */

        if (vides <= 0) {
            textEstat = game.add.text(game.world.centerX,
                                      game.world.centerY,
                                      'GAME OVER',
                                      { font: '30px Arial',
                                        fill: '#fff'
                                      });

            textEstat.anchor.setTo(0.5, 0.5);
            asteroides.callAll('kill');
            nau.kill();
        }

    }, null, this);
}
```

Si detectàvem una col·lisió nau - asteroide, restàvem una vida. Ara haurem d'afegir codi per veure si no ens queda cap vida i llavors amagar tots els elements gràfics del joc.

Per avisar que el joc s'ha acabat creem un nou element de text «GAME OVER» que es mostrarà al mig del joc, fent servir les propietats **game.world.centerX** i **game.world.centerY**.

Un cop mostrat tot el text, cal amagar tots els asteroides, això ho podem fer amb el mètode **calAll** del grup **asteroides** que ens permet indicar un callback a executar per cada element, en aquest cas el que volem és cridar al mètode **kill** de cada asterooide.

També cal amagar la nau, i per acabar fer que deixin de sortir més asteroides. Això ho podem fer modificant la pregunta que controla el temps de vida de l'asteroide i afegint el fet que la nau no estigui amagada (nau.alive)

CC BY

```
if (game.time.now > seguentAsteroide && nau.alive) {

    creaAsteroide();

}
```

També hauríem de preguntar si la nau està **alive** per disparar les bales.

CC BY

Step_11: Controlar el joc amb un dispositiu tàctil.

```
/* Variables pels controls tàctils*/
var touchDreta, touchEsquerra,
    cercleEsquerra, cercleDreta
    touchEsquerraPosInicial = new Phaser.Point(0, 0);

function ferCreate() {
    /* Step_11 */
    /* Defineix els callbacks pels dispositius mòbils*/
    if (game.device.desktop === false) {
        game.input.onDown.add(function (punter) {
            if (punter.pointerMode == 3 || punter.pointerMode ==
Phaser.PointerMode.CONTACT) {
                if (punter.positionDown.x < game.world.centerX && !
touchEsquerra) {
                    touchEsquerra = punter;
                    touchEsquerraPosInicial.copyFrom(punter.positionDown);

                } else if (punter.positionDown.x > 100
/*game.world.centerY*/ && !touchDreta) {
                    touchDreta = punter;
                }
            }
        }, this);
        game.input.onUp.add(function (punter) {
            if (punter.pointerMode == 3 ||
                punter.pointerMode == Phaser.PointerMode.CONTACT)
            {
                if (touchEsquerra && punter.id == touchEsquerra.id) {
                    touchEsquerra = null;
                    touchEsquerraPosInicial.setTo(0, 0);
                } else if (touchDreta && punter.id == touchDreta.id) {
                    touchDreta = null;
                }
            }

        }, this);
    }
}
```


CC BY

```
function ferUpdate() {

   /* Step_11 */
   /* Control tàctil*/
   if (game.device.desktop === false) {
      if (cercleEsquerra) {
         cercleEsquerra.destroy();
      }
      if (cercleDreta) {
         cercleDreta.destroy();
      }
      if (touchEsquerra) {
         cercleEsquerra = game.add.graphics(0, 0);
         cercleEsquerra.lineStyle(6, 0x00ff00);
         cercleEsquerra.drawCircle(touchEsquerraPosInicial.x,
                                   touchEsquerraPosInicial.y, 40);
         cercleEsquerra.lineStyle(2, 0x00ff00);
         cercleEsquerra.drawCircle(touchEsquerraPosInicial.x,
                                   touchEsquerraPosInicial.y, 60);
         /* Moviment de la nau*/
         var distancia = touchEsquerraPosInicial.distance(touchEsquerra,
                                                          true);
         if (distancia > 10) {
            var angRad = touchEsquerraPosInicial.angle(touchEsquerra);
            var angDeg = touchEsquerraPosInicial.angle(touchEsquerra, true);
            nau.rotation = angRad;
            game.physics.arcade.velocityFromAngle(angDeg,
                                                  distancia * 2,
                                                  nau.body.velocity);
         }
      } else {
         nau.body.acceleration.set(0);
         nau.body.angularAcceleration = 0;
      }
      if (touchDreta) {
         cercleDreta = game.add.graphics(0, 0);
         cercleDreta.lineStyle(6, 0xff0000);
         cercleDreta.drawCircle(touchDreta.x, touchDreta.y, 40);
         cercleDreta.lineStyle(2, 0xff0000);
         cercleDreta.drawCircle(touchDreta.x, touchDreta.y, 60);
         dispararBala();
      }
   }

}
```


CC BY

Step_11 - Explicació

A Phaser hi ha diferents punters per controlar tant el ratolí com els dispositius tàctils.

game.input.mousePointer: Sempre correspon al ratolí (si n'hi ha, és clar!)

game.input.activePointer: El darrer punter actiu (és a dir, que el que ha generat el darrer event)

game.input.pointer1 fins game.input.pointer10: Permet controlar fins a 10 punters per a jocs multi-touch. Phaser només inicialitza per defecte dos punters, si en necessitem més els hem d'iniciar natros. L'ordre dels punters s'assigna seqüencialment, 1 per la primera pulsació, i així succesivament. Si s'aixeca un dit el punter queda lliure i pot ser tornat a agafar al tornar a tocar el dispositiu.

Als punters poden associar-se callbacks pels següents events:

onDown: Al prémer el botó dret del ratolí o prémer un dispositiu tàctil. Només es dispara un cop, encara que premem la tecla durant una estona.

onUP: Al deixar de prémer el botó del ratolí o el dispositiu tàctil.

onTap: Es crida si es produeixen els events **down** i després **up** dins un període de temps prefixat.

onHold: Es crida si es manté premut durant una estona (bastant llarga).

Els callback reben un paràmetre punter que és un objecte amb tota la informació sobre la posició clicada, tipus de punter, identificador....

Tot i que en aquest exemple no ho farem servir, també es poden associar callbacks a events produïts sobre un sprite o sobre un botó. Els events als que podem associar callbacks són:

onInputOver

onInputOut

onInputDown

CC BY

onInputUp

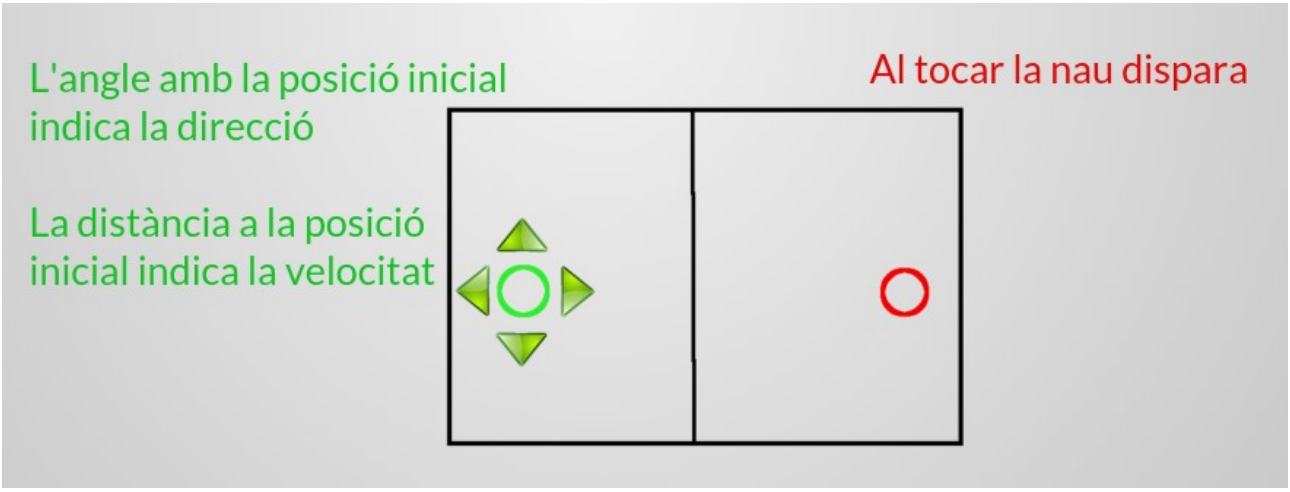
onDragStart

onDragStop

Si volem fer un joc que funcioni tant en ordinadors d'escriptori com en dispositius mòbils, hem de poder identificar en quin entorn ens trobem. Això ho podem fer amb la propietat **game.device.desktop** que val **true** quan és un PC i **false** si és un dispositiu mòbil.

També ens podem assegurar que el punter capturat correspongui a un punter tàctil i no a un ratolí. Això ho podem fer amb el mètode **pointerMode** del punter capturat. Els punters tàctils retornen el valor **Phaser.PointerMode.CONTACT**, tot que alguns dispositius mòbils tornen el valor 3, no documentat. D'aquesta forma ens podem assegurar que estem reaccionant davant d'una acció tàctil a un dispositiu mòbil.

La gestió dels controls tàctils la farem dividint la pantalla en dos zones, tal i com es mostra en aquest esquema:



Per controlar si ens han fet clic a la part dreta o esquerra de la pantalla, hem de verificar la coordenada x del punter rebut, el que farem serà desar la informació del punter capturat en dos variables: **touchDreta** i **touchEsquerra**. A més a més, en el cas que hagin tocat a l'esquerra, necessitem tenir la posició inicial, per poder calcular l'angle de la nau i la distància que ens indicarà la velocitat. Aquesta posició inicial la desarem a la variable **touchEsquerraPosInicial.**

CC BY

A la funció **ferCreate** l'únic que hem de fer és definir els callbacks **onDown** i **onUp** pels punters Phaser, verificar que siguin punters tàctils, i actualitzar les variables anteriors. Tota la lògica la posarem a la funció **ferUpdate**.

En aquesta funció hem de preguntar per les variables **touchDreta** i **touchEsquerra**, en el primer cas per disparar una bala, i en el segon cas per modificar la velocitat de la nau i la seva direcció en funció del punter tàctil. La forma implementada fa que el moviment de la nau sigui brusc, ja que canvia l'orientació de cop per adaptar-la a l'angle actual del punter esquerre. Es poden provar altres alternatives, però compliquen molt el codi i fan la nau més difícil de controlar. Aquesta aproximació dóna una jugabilitat acceptable, tot i que el moviment inicial de la nau sigui brusc.

Per donar pistes al jugador, dibuixarem un cercle en el lloc on s'hagi fet el touch. Si és a la part dreta (disparar) serà un cercle roig, i si és a la part esquerra (moviment) serà un cercle verd. Phaser disposa de funcions per dibuixar elements gràfics com línies i cercles. El primer que hem de fer és afegir al joc un element del tipus **Graphics** amb la funció **game.add.graphics**, indicant-li les coordenades (x, y) on volem posicionar el gràfic. Normalment indicarem com a referència l'origen del canvas, és a dir el punt (0, 0). Un cop creat aquest objecte podem modificar algunes de les seves propietats. Cridant al mètode **lineStyle** podem modificar el gruix de la línia a dibuixar, el color del pinzell i opcionalment la transparència. Un cop definits aquests paràmetres podem cridar a la funció **drawCircle** indicant-li les coordenades del centre de la circumferència i el seu radi.

Per modificar la velocitat de la nau, ens assegurarem que l'usuari hagi fet un touch en la part esquerra, i a més a més que hagi fet un moviment amb el dit, per això només modificarem la velocitat i la direcció si la distància entre el touch actual i el touch inicial és superior a 10 pixels. Aquesta distància la calculem amb el mètode **distance** de l'objecte **Point** on hem desat el touch inicial. Li hem d'indicar l'altre punt (el touch actual) i si volem arrodonir la distància (true).

Si necessitem actualitzar la velocitat i la direcció de la nau, necessitem saber l'angle entre el touch inicial i el touch actual. Això ho fem amb el mètode **angle** de l'objecte **Point**. Li hem d'indicar l'altre punt i un boolea opcional per indicar si volem que ens torni l'angle en radians o en graus. Necessitarem totes dos dades, així que cridarem a aquest mètode dos cops.

CC BY

La direcció de la nau (propietat **rotation**) l'establirem al mateix angle calculat en radians, mentre que la velocitat de la nau la calcularem amb la funció **game.physics.arcade.velocityFromAngle** que necessita saber l'angle en graus, la velocitat (serà igual a la distància calculada en pixels i multiplcada per dos, per tal que sigui més responsiu) i la propietat on volem desar el resultat calculat (**nau.body.velocity**).

Per acabar, quan es deixi de tocar la part esquerra posarem a zero les acceleracions per tal que el fregament acabi aturant la nau.

CC BY

Step_12: Afegir sons al joc

Cal afegir música de fons, so a l'accelerar i so en les explosions.

```
/* Sons del joc*/
var musica_sound, accelera_sound, explota_sound;

function preload() {
    /* Carrega els sons*/
    game.load.audio('musica', 'assets/musica.ogg');
    game.load.audio('accelera', 'assets/thrust.ogg');
    game.load.audio('explosio', 'assets/explosio.ogg');
}

function create() {
    /* Crea els elements d'audio */
    musica_sound = game.add.audio('musica', 0.5, true);
    accelera_sound = game.add.audio('accelera', 1, true);
    explosio_sound = game.add.audio('explosio');

    /* Step_3 */
    /* Controla les tecles que gestionen el moviment de la nau */
    acelerar = game.input.keyboard.addKey(Phaser.Keyboard.UP);
    acelerar.onDown.add(function () {
        if (!accelera_sound.isPlaying) {
            accelera_sound.play();
        }

    });
    acelerar.onUp.add(function () {
        accelera_sound.stop();
    });
    /* Inicia la reproducció del fons musical */
    musica_sound.play();
}

function ferUpdate() {
    /* Step_7 */
    /* Revisa les colisions entre el grup de bales i el grup d'asteroides */
    game.physics.arcade.overlap(bales, asteroides,
                function (bala, asteroide) {
                explosio_sound.play();
            },
    null, this);

    /* Hem de fer el mateix al Step_8 */

}
```


CC BY

Step_12 - Explicació

Phaser permet carregar diferents tipus de fitxers d'audio: mp3, ogg... Ho fa mitjançant l'objecte **Sound**, que permet reproduir audio. Al crear l'objecte hem d'indicar si volem que reprodueixi l'arxiu de forma contínua en un bucle (loop) o només es reprodueixi un cop.

Els principals mètodes de l'objecte Sound són

play i stop: Inicia o atura la reproducció

pause i resume: Pausa o segueix la reproducció

restart: Resitua el punter de l'arxiu a l'inici

fadeIn i fadeOut: Inicia o atura la reproducció amb un increment/decrement progressiu del volum

Les principals propietats de l'objecte Sound són:

volume: Propietat que permet ajustar el volum de reproducció
mute: Permet silenciar o donar volum a la reproducció

A la funció **ferPreload** hem de carregar els arxius d'audio i associar-los una clau. Ho fem amb la funció **game.load.audio**, i després a la funció **ferUpdate** afegim l'audio al joc amb la funció **game.add.audio** que ens retorna un objecte **Sound** que ens permetrà iniciar i aturar la reproducció de la música. Aquesta funció necessita la clau de l'element audio a carregar, i també li podem indicar el volum de reproducció i si volem que l'arxiu d'audio es repeteixi contínuament en un bucle.
Un cop els objectes sonors creats, només cal invocar els seus mètodes **play** per iniciar la reproducció. En el cas de l'acceleració, si ja tenim la tecla premuda i estem accelerant, si tornem a fer play l'audio es reinicia, així que només hem de fer **play** quan el so encara no estigui reproduint-se. Això ho podem determinar amb la propietat **isPlaying** de l'objecte **Sound**.

Per aturar el so hem de cridar al mètode **stop** de l'objecte **Sound**.

CC BY

To Do....

- Mostrar les vides com un grup de imatges de la nau
- Afegir una vida al fer un cert nombre de punts
- En acabar, permetre reiniciar el joc prement una tecla
- Mostrar una pantalla de benvinguda a l'inici del joc
- En acabar, aturar les accions tàctils, música...
- Crear diferents gràfics pels asteroides i mostrar-los
- Al tocar un asteroide gros mostrar-de dos de petits
- Incrementar progressivament la dificultat
- Registrar les puntuacions màximes
- ...

CC BY

Instal·lació de l'entorn de desenvolupament

Per instal·lar el software necessari necessitem un terminal Linux per poder posar les comandes necessàries.

Actualitzarem els repositoris de software per tal que ens trobi les darreres versions del software a instal·lar.

```
$sudo apt-get update
```

Per descarregar els arxius de codi pel tutorial podem fer servir l'eina git.

```
$sudo apt-get install git
```

Ara copiem el repositori on hi ha tot el codi d'exemple i els elements gràfics necessaris.

```
$git clone https://github.com/jaumeramos/asteroids
```

Per instal·lar el software, ens farà falta l'eina curl que ens permetrà descarregar arxius des de la web.

```
$sudo apt-get install curl
```

Després instal·larem l'eina node.js, que ens permetrà executar programes javascript des del nostre equip. Per fer-ho hem d'afegir un repositori de software i després instal·lar el paquet nodejs.

```
$curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -
$sudo apt-get install -y nodejs
```

Ara instal·larem el servidor web http-server. La forma més senzilla d'instal·lar-lo és amb l'eina d'empaquetat npm, que primer haurem d'instal·lar.

```
$curl -sL https://npmjs.org/install.sh | sudo -E bash -
```

Un cop fet ja podem utilitzar npm per instal·lar el servidor web.

```
$sudo npm install http-server -g
```

Iniciem el servidor web sobre la carpeta que conté el codi del joc amb la comanda:

```
$http-server asteroids/code
```


CC BY

Ens indica que ha configurat un servidor web al port 8080, i a la pantalla ens mostrarà les peticions fetes. Podem comprovar el funcionament anant obrint el navegador i posant l'adreça http://localhost:8080 també podem fer ctrl+clic sobre els enllaços que ens mostra la terminal amb l'adreça del servidor web creat.

Per editar el codi farem servir el IDE Brackets, fet amb JavaScript i que està orientat al desenvolupament web, disposant de moltes eines que faciliten la tasca de crear pàgines web i programes JavaScript. Hi ha un paquet .deb per les distribucions derivades de Debian, com Ubuntu i Mint. Haurem d'obrir un nou terminal, ja que en l'anterior tenim funcionant el servidor web.

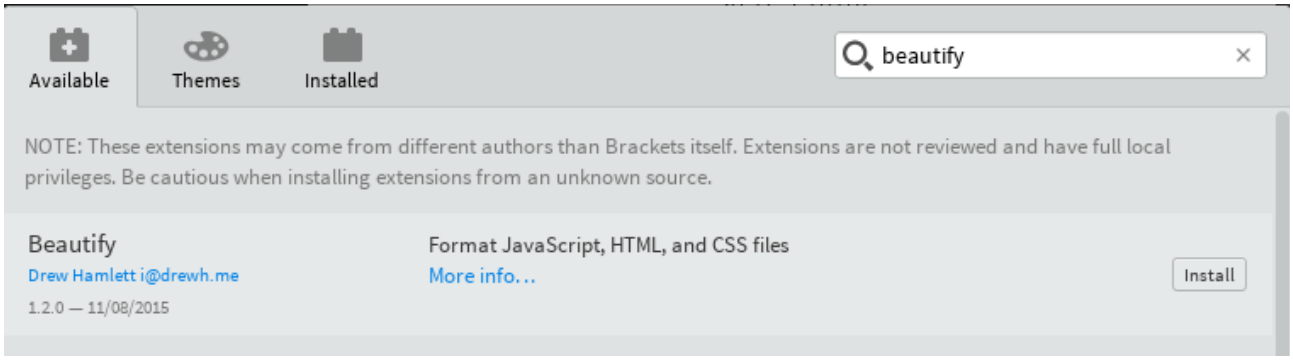
```
$curl -L -o brackets.deb
https://github.com/adobe/brackets/releases/download/release-
1.6%2Beb4/Brackets.1.6.Extract.64-bit.deb

$sudo dpkg -i brackets.deb
```

Per posar en marxa l'editor ho podem fer des de la mateixa terminal.

```
$brackets &
```

Obrim la carpeta asteroids/code on tenim el codi d'exemple i... ja podem començar!!!!

A Brackets hi ha algunes extensions totalment recomanables. La primera es diu Beautify i permet formatar automàticament el codi. Per instal·lar extensions hem d'anar a File/Extension Manager i cercar pel nom. Un cop trobada la extensió fem clic al botó install.



Un altra extensió molt recomanable és «Paste and Indent», de Andrew Huth (https://github.com/ahuth) que permet indentar el codi enganxat automàticament.

I un tema interessant per la forma de ressaltar el codi és «Delkos Dark Theme».

CC BY

Brackets és un editor pensat pel desenvolupament web, i té funcions molt interessants, per exemple, us mostra on falten tags html per tancar, quan voleu tancar un tag ell automàticament us tanca el darrer obert. Si poseu el cursor damunt d'un color, veureu en un requadre aquest color, i de la mateixa forma, si poseu el cursor damunt una imatge veureu un snapshot d'aquesta. També permet editar «inline» el codi css de qualsevol element, si feu ctrl+E quan esteu en un element css us «obrirà» un quadre on podreu editar el css associat, encara que estigui en altres arxius.

Una funcionalitat molt interessant les el live-preview, que es troba a la part superior dreta, en la barra d'eines vertical. Permet previsualitzar els canvis fets en el mateix moment que els fem, simplificant molt el desenvolupament. Aquesta funcionalitat ara només es pot utilitzar amb el navegador chrome.

CC BY

Comandes per instal·lar l'entorn de desenvolupament.

```
sudo apt-get update

sudo apt-get install -y git

git clone https://github.com/jaumeramos/asteroids

sudo apt-get install -y curl

curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -

sudo apt-get install -y nodejs

curl -sL https://npmjs.org/install.sh | sudo -E bash -

sudo npm install http-server -g

curl -L -o brackets.deb
https://github.com/adobe/brackets/releases/download/release-
1.6%2Beb4/Brackets.1.6.Extract.64-bit.deb

sudo dpkg -i brackets.deb

rm brackets.deb

brackets &

http-server asteroids/code
```


CC BY

Pàgines on trobar informació de les eines utilitzades.

Node.js: Programa per executar aplicacions JavaScript fora del navegador.

https://nodejs.org/en/download/

Gestor de Paquets NPM.

https://www.npmjs.com/

Servidor Web fet en JavaScript i que fa servir node.js

https://www.npmjs.com/package/http-server

Sistema de control de versions GIT

https://git-scm.com/

Editors HTML WYSIWYG OnLine

http://htmleditor.in/index.html

http://www.html.am/html-editors/online-html-editor.cfm

IDE Brackets per desenvolupament Web

http://brackets.io/

Framework PHASER per desenvolupar jocs

http://phaser.io/

Eina per fer presentacions en HTML5

http://strut.io/

Llibreria JavaScript per fer efectes semblants a Prezi en presentacions HTML5

https://github.com/impress/impress.js/

CC BY