

## E. Guía de desarrollo

Una de las cosas que se tuvo en cuenta a la hora de desarrollar el juego fue estructurar todos sus componentes de forma clara para que en un futuro se pudieran expandir sus funcionalidades o agregar nuevos tipos de operaciones sin que surgieran dificultades. El presente anexo pretende ser una pequeña guía para orientar al desarrollador responsable de la expansión.

### E.1. Añadir un nuevo tipo de enemigo

Lo primero que hay que hacer en este caso es crear un nuevo componente para el nuevo enemigo, y alojarlo en la carpeta donde se encuentran los otros componentes de enemigos. Para hacer esto basta con copiar y pegar uno de los existentes para después modificarlo:



Figura 34: Estructura del nuevo enemigo

Como se observa en la figura 34, creamos el componente con el nombre que queramos para el nuevo enemigo, para posteriormente crear una función init. La función init es la que toma CraftyJS por defecto cada vez que se crea una entidad de este tipo.

A esta función le indicamos que requerimos el componente Enemy y su hoja de sprites para que los cargue “spr\_enemy1”. Para crear una animación, la definimos con this.reel(). Los parámetros indican en qué casilla empieza el primer sprite, la velocidad y el número de sprites (Casilla 0x0, 300ms de velocidad por los 3 sprites, 100ms cada sprite).

El componente “spr\_enemy1” que contiene los gráficos del enemigo lo definimos en Components.js, que precarga todos los gráficos. En este archivo veremos dos vectores de imágenes. Simplemente añadimos el nombre del fichero dentro y modificamos el código de las funciones para incluirlo.

```
var EDITOR_ICONS = ['char.png', 'floor1.jpg', 'floor2.jpg', 'floor3.jpg', 'floor4.jpg',  
                  'floor5.jpg', 'abyss.jpg', 'hide.png', 'chest.png', 'enemy1.png',  
                  'enemy2.png', 'enemy3.png', 'enemy4.png', 'enemy5.png', 'exit.jpg'];  
var GAME_ICONS = ['char.png', 'floor1.jpg', 'floor2.jpg', 'floor3.jpg', 'floor4.jpg',  
                 'floor5.jpg', 'abyss.jpg', 'hide.png', 'chest.png', 'enemy1.png',  
                 'enemy2.png', 'enemy3.png', 'enemy4.png', 'enemy5.png', 'exit.jpg',  
                 'multiplayer.png'];
```

Figura 35: Vectores de sprites

Para incluir a este nuevo enemigo en el editor, tendremos que editar drawTile(x,y,type) en “engine.js” y “enemy.js” para incluir nuestro nuevo tipo de entidad en la paleta de pinceles y en el componente padre, respectivamente.

Por último, queda integrar la lógica de las batallas en el módulo “battle.js”, que queda a cargo del desarrollador. Si la operación tiene que ver con polinomios hay un amplio surtido de parsers de polinomios y términos, así como generadores de vectores y código html asociado a los mismos.

## E.2. Añadir efectos y música

Para realizar esto únicamente hay que modificar el módulo Audio. Introducimos el nombre del sonido en el vector de ficheros de audio y creamos una función para reproducirlo en el caso de ser un efecto o lo introducimos en la función playLevel() si es una canción.

## E.3. Añadir terrenos

El proceso es prácticamente idéntico al de crear un enemigo. Hay que crear el componente en la carpeta de terrenos y hacer las modificaciones explicadas en el apartado E.1.

## E.4. Modificación de parámetros

A continuación se detallan una serie de parámetros que afectan a la mecánica del juego y cómo modificarlos:

- Poder de curación de los bonus de vida, así como el número de bonus que se entregan por cofre: Editar “bonus.js” y cambiar las propiedades que aparecen en la zona privada del módulo.
- Cambiar el número de términos máximos de un polinomio, coeficiente máximo, probabilidad de que un término sea +/- y probabilidad de término nulo: Editar propiedades de “battle.js”.
- Cambiar la cantidad de daño recibido por golpe tanto para escudo como para daño normal: Editar propiedades de “damage.js”.
- Cambiar radio de detección: Editar la variable siguiente en “detection.js”:

```
var detectionRange = Crafty.map.search({_x: this.x-320, _y: this.y, _w: 640, _h: 32}, true);
```

Donde 640 son 10 casillas, ya que cada casilla son 32 píxeles (como se explicó en la página 19) y el radio busca en dos direcciones, luego:  $10 \cdot 32 \cdot 2 = 640$ . Al buscar en 2 direcciones tenemos que centrarlo, por eso a la propiedad de entrada `_x` es la mitad, 320.

- Cambiar el aspecto de las aplicaciones de lava: Cambiar los parámetros de la propiedad “drawLava” en “lava.js”. No se recomienda incrementar el número de partículas por la sobrecarga de CPU que genera.
- Cambiar la gravedad o la altura del salto: Editar la constante de gravedad proporcionada a `this.gravity().gravityConst()` y la propiedad “JUMPY” en “movement.js”.
- Cambiar la velocidad de patrulla de los enemigos: Pese a que no se recomienda ya que una velocidad excesiva puede hacer que el algoritmo de patrulla funcione

incorrectamente, basta con modificar la propiedad `this._patrolSpeed` de “patrol.js”.

### 6.1.1. Modificación del gestor de alumnos

Para implementar nuevas opciones es suficiente con incluir la nueva lógica del gestor en la carpeta de contenido público, en los ficheros “view.js” y “controller.js”. Puede ser necesario editar y ampliar el modelo de datos para adaptarse a las nuevas funcionalidades.

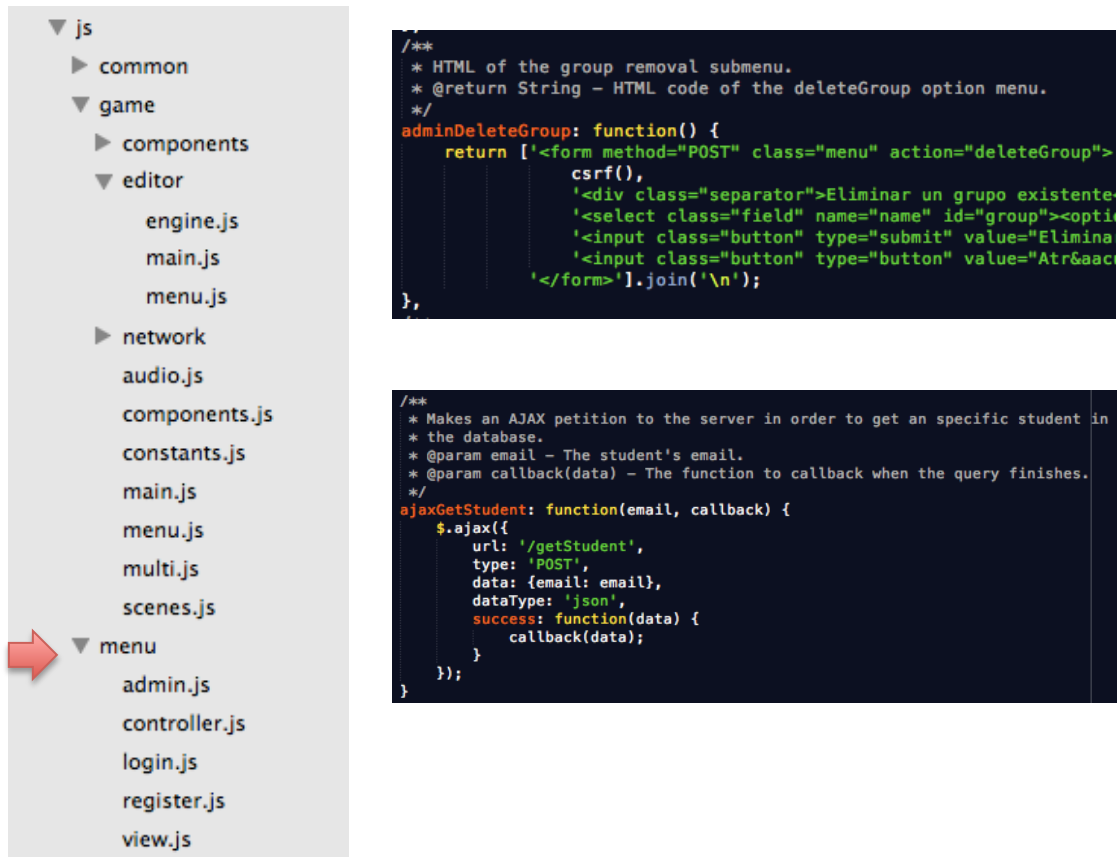


Figura 36: Rutas (Izquierda), Vista (Arriba), Controlador (Abajo)

## G. Guía de despliegue en Heroku

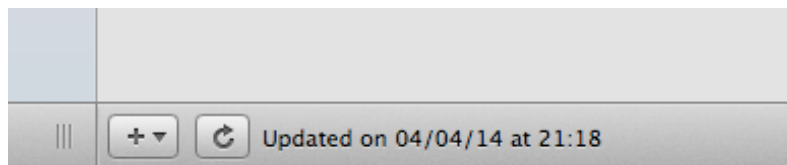
---

En algunos casos puede ser interesante la opción de subir el juego a Internet para jugar desde cualquier sitio y no sólo en las aulas. Heroku es un servicio en la nube que permite alojar aplicaciones creadas con node.js entre otros muchos tipos más. Se ha elegido este servicio por contar con un plan gratuito y por ser sencillo de configurar.

### G.1. Creación de un repositorio en GitHub

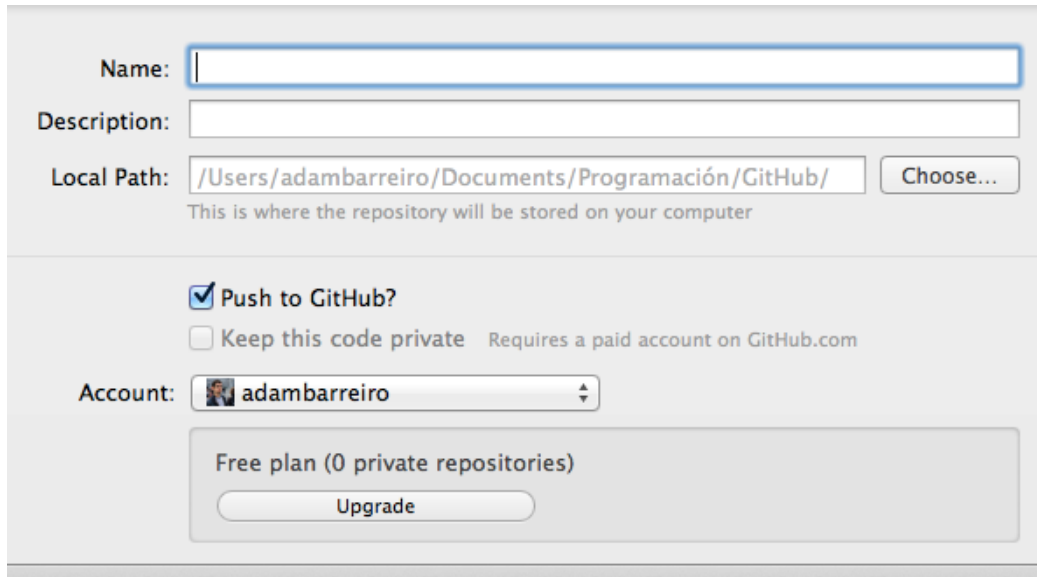
El primer requisito para subir el juego a Heroku es tener alojado el código en este servicio gratuito de control de versiones. Para ello iremos a <http://www.github.com> y crearemos una cuenta rellenando los datos que nos solicita. Posteriormente nos descargaremos su programa de escritorio y lo instalaremos.

Cuando tengamos GitHub instalado, crearemos un repositorio nuevo con el botón de la figura 60, que aparece en la parte inferior de la aplicación con un símbolo “+”:



*Figura 60: Creamos un repositorio con el botón “+”*

Al hacer click nos saldrá un formulario como el de la figura 61. Aquí se nos pide el nombre del repositorio, una breve descripción (opcional) y la carpeta donde lo alojaremos localmente. Aquí es donde debemos introducir el proyecto una vez creamos el repositorio.



The screenshot shows the GitHub repository creation interface. It includes a 'Name' field, a 'Description' field, and a 'Local Path' field with a 'Choose...' button. Below these, there are checkboxes for 'Push to GitHub?' (checked) and 'Keep this code private'. An 'Account' dropdown menu is set to 'adambarreiro'. At the bottom, it shows 'Free plan (0 private repositories)' with an 'Upgrade' button.

Figura 61: Creación de un repositorio en GitHub

Si lo hemos creado correctamente, en nuestra cuenta de GitHub aparecerá el repositorio (figura 62).

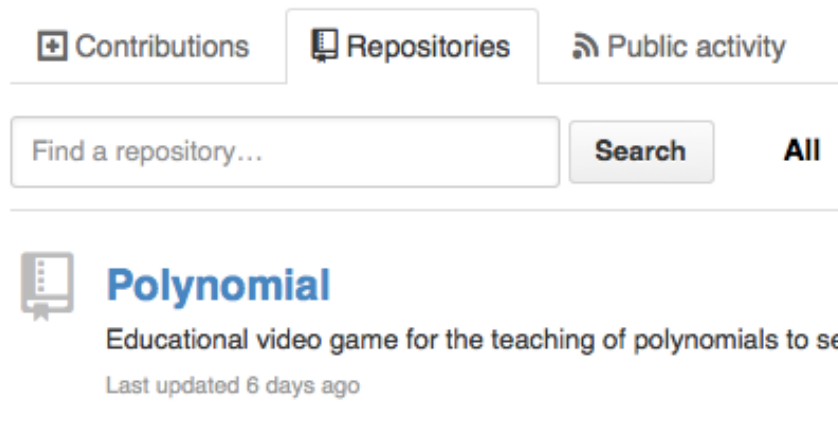


Figura 62: Repositorios en GitHub

Al introducir el proyecto en la carpeta local que hemos indicado al crear el repositorio, deberemos sincronizarlo para subir los cambios. Para esto activamos la opción Commit & Sync con el botón verde de la figura 63 y pulsamos el botón.

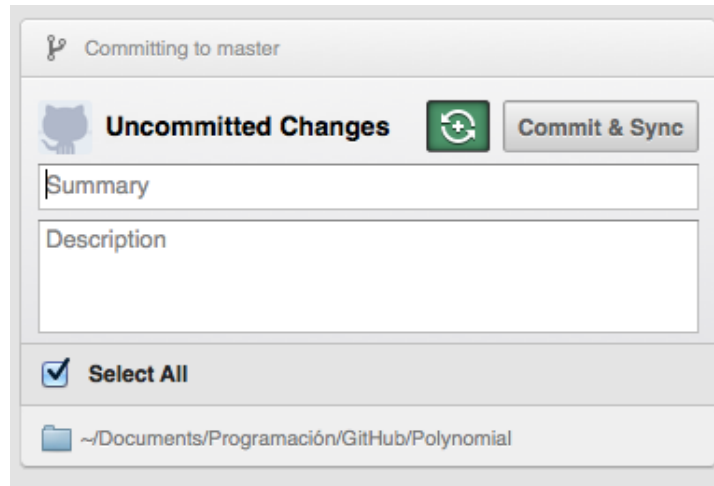


Figura 63: Cómo hacer un commit en GitHub

## G.2. Creación de una base de datos remota

Para subir la aplicación a Internet también es necesario alojar la base de datos en un entorno online, de modo que necesitamos registrarnos en un servicio que ofrezca una base de datos MongoDB. Este sitio es <http://www.mongolab.com>, que se ha elegido por contar con un modo gratuito sencillo. Lo único que necesitamos es una cuenta y crear una nueva base de datos, teniendo en cuenta que debemos elegir las opciones de la figura 64.

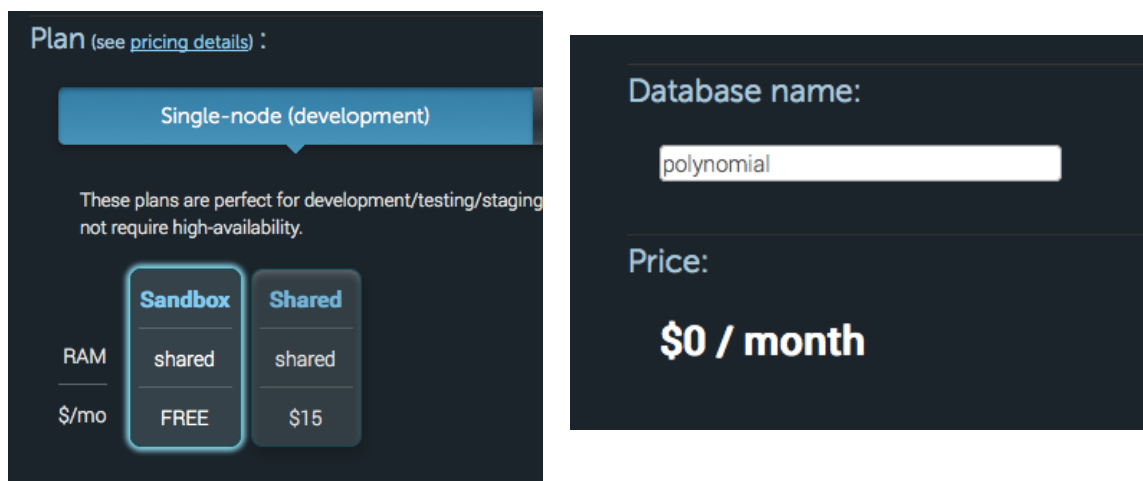


Figura 64: Configuración de MongoLab

Una vez creada, debemos asignarle un usuario y contraseña para evitar el acceso de personas extrañas. Vamos a la sección de usuarios y rellenamos el formulario de la figura 65 con el usuario y contraseña que queramos.

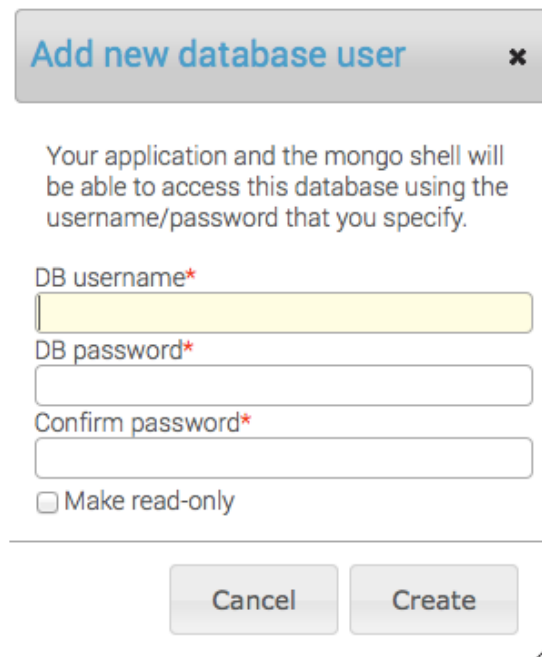


Figura 65: Creación de usuarios en MongoLab

Todos estos campos son los que pondremos en el fichero “database.txt” dentro de la carpeta “configuration/heroku” como se verá en el apartado G.4. En la página principal de MongoLab ya nos dan el contenido del fichero, que sigue esta sintaxis:

```
mongodb://<usuario>:<password>@*.mongolab.com:nnnnn/<nombrebdd>
```

A continuación tenemos que abrir una consola de comandos y ejecutar “mongodump -d polynomial -o directorio”. Esto nos dará una copia exacta de nuestra base de datos que podremos importar en MongoLab con otra instrucción que también tenemos que ejecutar y que MongoLab nos proporciona en el apartado de “Tools” y que aparece en la figura 66.

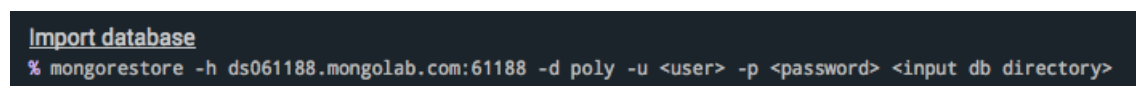


Figura 66: Instrucción de MongoLab para importar BBDD



### G.3. Instalación del Heroku Toolbelt

Procedemos a registrarnos en Heroku para posteriormente descargarnos su software, que nos permitirá gestionar las aplicaciones que subamos mediante línea de comandos. Este software está accesible desde <https://toolbelt.heroku.com> y se llama Heroku Toolbelt.

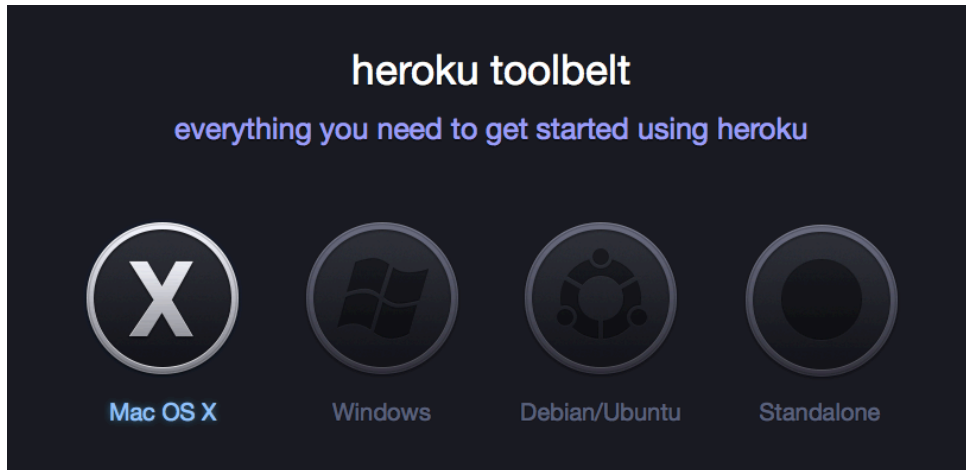


Figura 67: Pantalla de descarga de Heroku Toolbelt

Una vez instalado, abrimos una consola de comandos y hacemos login mediante la instrucción de la figura 68.

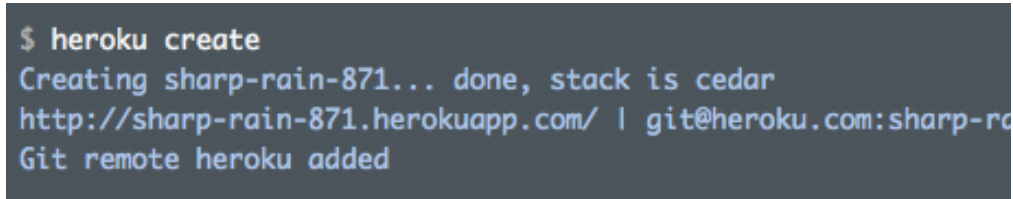
```
$ heroku login
Enter your Heroku credentials.
Email: zeke@example.com
Password:
Could not find an existing public key.
Would you like to generate one? [Yn]
Generating new SSH public key.
Uploading ssh public key /Users/adam/.ssh/id_rsa.pub
```

Figura 68: Login en Heroku

Para que los servidores de Heroku sepan qué fichero tienen que ejecutar, tenemos que introducir en la carpeta del proyecto un fichero llamado Procfile con este contenido:

```
web: main.js
```

Este fichero ya está incluido en la carpeta, de modo que no hace falta crearlo otra vez. Para crear una aplicación tenemos que abrir una consola de comandos y movernos a la carpeta donde tenemos el código en local, para posteriormente ejecutar el comando de la figura 69.



```
$ heroku create
Creating sharp-rain-871... done, stack is cedar
http://sharp-rain-871.herokuapp.com/ | git@heroku.com:sharp-rain-871
Git remote heroku added
```

*Figura 69: Crear una aplicación en Heroku*

Opcionalmente podemos introducir un nombre detrás de “create” para ponerle un nombre a nuestra aplicación, si no Heroku le asignará un nombre aleatorio, como “sharp-rain-871”:

```
$ heroku create pepito-app
```

### G.4. Modificación de código y configuración

Antes de subir la aplicación debemos asegurar que la configuración es correcta. Para ello nos dirigimos a la carpeta del proyecto. Dentro veremos otra carpeta llamada “configuration” con varios ficheros de texto y dos carpetas dentro, “heroku” y “local”. Si borramos los ficheros de texto y copiamos ahí mismo los que están dentro de “heroku” el juego debería funcionar. No obstante comprobamos su contenido:

- El contenido de “database.txt” debería ser del estilo `mongodb://<usuario>:<password>@*.mongolab.com:nnnnn/<nombrebdd>` como se ha comentado en el apartado G.2.
- El contenido de “http\_port.txt” debería ser “5000”.
- El contenido de “https\_port.txt” es indiferente porque Heroku no soporta HTTPS en su forma gratuita.
- El contenido de “groups\_needed.txt” es indiferente, podemos poner “false” para no requerir la asignación del administrador si estamos en fase de pruebas de algún tipo.

Heroku implementa WebSockets de forma experimental, y de un tipo muy específico, por lo que también debemos modificar una línea de código concreta, que se detalla en la figura 70.

```

/**
 * Creates the socket.io structures and message protocol for the multiplayer.
 */
function createRouter(server) {
  // Creates the socket
  io = io.listen(server);
  io.configure( function(){
    io.set('log level', 1);
    // heroku labs:enable websockets -a myapp
    // Local:
    // io.set('transports', ['websocket', 'xhr-polling', 'flashsocket', 'htmlfile', 'jsonp-polling']);
    // Heroku:
    // io.set('transports', ['xhr-polling', 'websocket', 'flashsocket', 'htmlfile', 'jsonp-polling']);
    io.set('transports', ['websocket', 'xhr-polling', 'flashsocket', 'htmlfile', 'jsonp-polling']);
    io.set('polling duration', 10);
  });
  // Connection events
  io.on('connection', function (socket) {
    responseReady(socket);
    responseJoin(socket);
    responseEngaged(socket);
    responseClose(socket);
    responseDisconnect(socket);
    responseSender(socket);
  });
}
exports.createRouter = createRouter;

```

Figura 70: Modificación de código requerida para Heroku

Como se puede observar, si vamos a subir la aplicación a Heroku hay que usar la línea de código que hace un `io.set()` con “xhr-polling” en primer lugar, de modo que sustituiremos la línea que está activa por ésta. Además, en la consola de comandos ejecutaremos:

```
$ heroku labs:enable websockets -a nombre_de_mi_aplicacion
```

## G.5. Despliegue

Una vez configurado, volvemos a la consola de comandos. Nos desplazamos a la carpeta del proyecto y ejecutamos la instrucción de la figura 71. Si todo va bien, al introducir la url que se nos indica en la última línea de texto deberíamos ver el juego ejecutándose.

```
$ git push heroku master
Counting objects: 343, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (224/224), done.
Writing objects: 100% (250/250), 238.01 KiB, done.
Total 250 (delta 63), reused 0 (delta 0)

-----> Node.js app detected
-----> Resolving engine versions
      Using Node.js version: 0.10.3
      Using npm version: 1.2.18
-----> Fetching Node.js binaries
-----> Vending node into slug
-----> Installing dependencies with npm
      ....
      Dependencies installed
-----> Building runtime environment
-----> Discovering process types
      Procfile declares types -> web

-----> Compiled slug size: 4.1MB
-----> Launching... done, v9
      http://sharp-rain-871.herokuapp.com deployed to Heroku

To git@heroku.com:sharp-rain-871.git
 * [new branch]      master -> master
```

*Figura 71: Despliegue de la aplicación en Heroku*