Programación

Decimoquinta semana

Febrero 2022

Cruz García, Iago

Anotaciones previas

Estos ejercicios son para familiarizarse con el lenguaje, la sintaxis y cómo resolverlos. Los primeros son sencillos y se va incrementando la dificultad. A continuación se presentan una serie de instrucciones que son necesarias para la resolución de los ejercicios:

- **alert(*parámetro*):** esta instrucción permite mostrar por pantalla un cartel con texto para mostrar la solución de algunos ejercicios.
- **console.log(*parámetro*):** esta instrucción permite mostrar en consola (F12 en el navegador) la solución de algunos ejercicios o trazar el código para comprobar que todo se ejecuta correctamente.
- **prompt(*texto, ejemplo*):** Muestra en pantalla un recuadro de **texto** y un cuadro para introducir texto con un **ejemplo**.
- Para poder ejecutar código JavaScript en Visual Studio Code debéis crear un fichero JavaScript (miScript.js) y un HTML básico (index.html por ejemplo) y dentro de la etiqueta <head> escribir los siguiente:
    - <script src="miScript.js"></script> comillas incluidas
- Ahora que sabemos encapsular creando funciones o métodos, se pueden hacer los ejercicios en el mismo fichero, simplemente comentando las llamadas a métodos que no necesiteis.

```
ejercicio_1()
//ejercicio_2()
//ejercicio_3()
```

Ejercicios

IMPORTANTE: A partir de ahora algunos ejercicios deben hacerse en múltiples ficheros .js, por lo que en vez de entregar todos en un mismo main.js, será necesario dividirlos en directorios. Se aconseja la estructura de "PrácticaX_ejercicio1" y dentro el index.html y los ficheros .js necesarios.

Para estos ficheros, lo mejor es agrupar aquellas funciones o métodos que realicen tareas similares (entradas.js o salidas.js por ejemplo). En caso de duda, no importa que un método quede aislado en un fichero.

El fichero que realice las llamadas a los métodos, que aune toda la funcionalidad, debe nombrarse como main.js y no debe tener más que un método que se llame igual y una llamada a este mismo.

Ejercicio 0

Vamos a utilizar como base el ejercicio modificado del juego por turnos que hicimos en el tema de clases y modificamos con JQuery. Vamos a otorgarle interactividad y darle una vuelta para que se haga algo más entretenido.

Ejercicio 1

Añade un botón al HTML (con la etiqueta <button>). Asígnale un identificador (id). Como ya tenemos un método que permite al héroe atacar cada turno, genera un evento que enlace el ataque con el click del botón. En los apuntes de esta semana encontrarás cómo hacerlo.

Ejercicio 2

Ahora hay que modificar el bucle jugable. Al no ser automático, deja de tener sentido un bucle while en el que se turnan para golpearse los personajes, si no que el enemigo debería responder a las acciones del usuario.

Para ello, comenta el bucle que hicimos para los turnos y en el evento creado para atacar, añade al final la acción del enemigo.

Ejercicio 3

Nuestro juego ya tiene más interactividad que antes, pero sería interesante añadirle alguna opción de juego para hacer alguna elección.

Vamos a añadir primero a los atributos de los personajes uno que sea "defensa" y otro que sea "isDefendiendo". El primero tendrá un valor numérico, mientras que el segundo será un valor booleano que nos servirá para el siguiente paso.

Ejercicio 4

Teniendo las variables necesarias, debemos modificar el método de recibir daño, de tal manera que si el personaje está defendiendo, es decir, la variable isDefendiendo es igual a *true*, el daño recibido se reduce por el valor que tenga el atributo defensa recién creado. Si no, el daño es el mismo que antes.

Ejercicio 5

Lo que debemos hacer a continuación es un botón en el documento HTML con un identificador para indicar que queremos defendernos ese turno. Además, debemos generar un evento en el que se modifique el valor de de "isDefendiendo" ese turno, de tal manera que el enemigo al atacar haga menos daño como establecimos en el apartado anterior.

Ejercicio 6

Ya casi tenemos el juego preparado. Ahora vamos a programar el continuar o terminar partida.

Hacer una función que pare la partida es complicado, JavaScript por naturaleza es un lenguaje asíncrono, lo que quiere decir que se pueden ejecutar funciones fuera del flujo del programa sin interrumpirse unas a otras. Esto es una ventaja en algunos casos ya que nos permitirá utilizar AJAX sin interrupciones ni esperas, pero una desventaja cuando queremos que la ejecución cese. Por lo que si quisieramos agregar un botón de "continuar" o "terminar partida", los botones de atacar y defender no se desactivarían, dando lugar a errores. No solo eso, si no que mostrarlos y ocultarlos puede ser tedioso.

Para ello, vamos a añadir en la muerte del enemigo un mensaje con prompt que pregunte si queremos continuar o terminar. Depende de lo que devuelva el método, se generará un enemigo o se terminará la partida.

En el héroe hacer lo mismo, pero permitiendo volver a introducir un nombre (o los atributos necesarios) como si fuera la primera vez que carga la página.

Pista: Básicamente se debe de modificar los métodos que indican cuando muere un héroe o un enemigo.

Extra

Estas son algunas ideas que podéis implementar pero requieren romperse un poco la cabeza o hacer cambios triviales que no contemplo como ejercicios:

- Añadir la opción de "magia" para que pueda utilizar hechizos el jugador y solo sea posible si tiene energía para ello.
- Añadir la opción de subir de nivel y elegir que atributos subir de nivel. Este apartado es relativamente complejo por lo comentado en el ejercicio 6.
- Aleatorizar el comportamiento del enemigo. Hasta ahora tras los eventos de atacar o defender el enemigo siempre ataca. Probad a modificarlo de tal manera que la acción que haga a continuación sea aleatoria.
- En vez continuar o terminar partida, añadir un botón de "buscar enemigo" que solo aparezca o funcione cuando se derrota al ya existente.