Entrega 3 - Juego de disparos

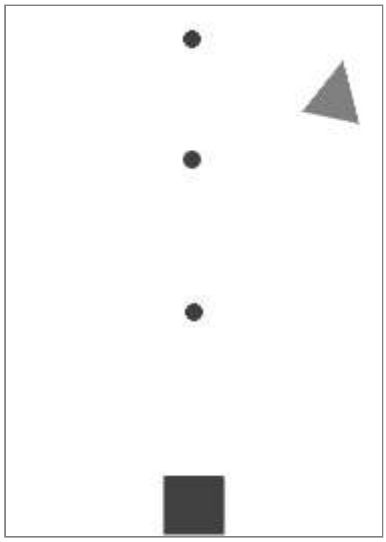
Versión: 18 de Febrero de 2020

Objetivo

Practicar con clases, Booleans, Strings y con el manejo de eventos.

Descripción de la práctica

En esta entrega vamos a desarrollar un juego completo usando HTML, CSS y JavaScript. El juego consiste en un juego clásico de disparos, en el que manejaremos a nuestro personaje (cuadrado) utilizando las flechas del teclado o la pantalla táctil. El objetivo del juego es disparar a una serie de formas que aparecerán en la pantalla para convertirlas en estrellas, a la vez que esquivamos sus disparos. Para comenzar el desarrollo partimos de la versión básica del juego cuyo código proporcionamos.



Descargar el código del proyecto

El proyecto se descarga en el ordenador local con estos comandos:

El proyecto debe clonarse en el ordenador desde el que se está trabajando

```
$ git clone https://github.com/CORE-2020/Entrega3_juego
```

Entrar en el directorio de trabajo

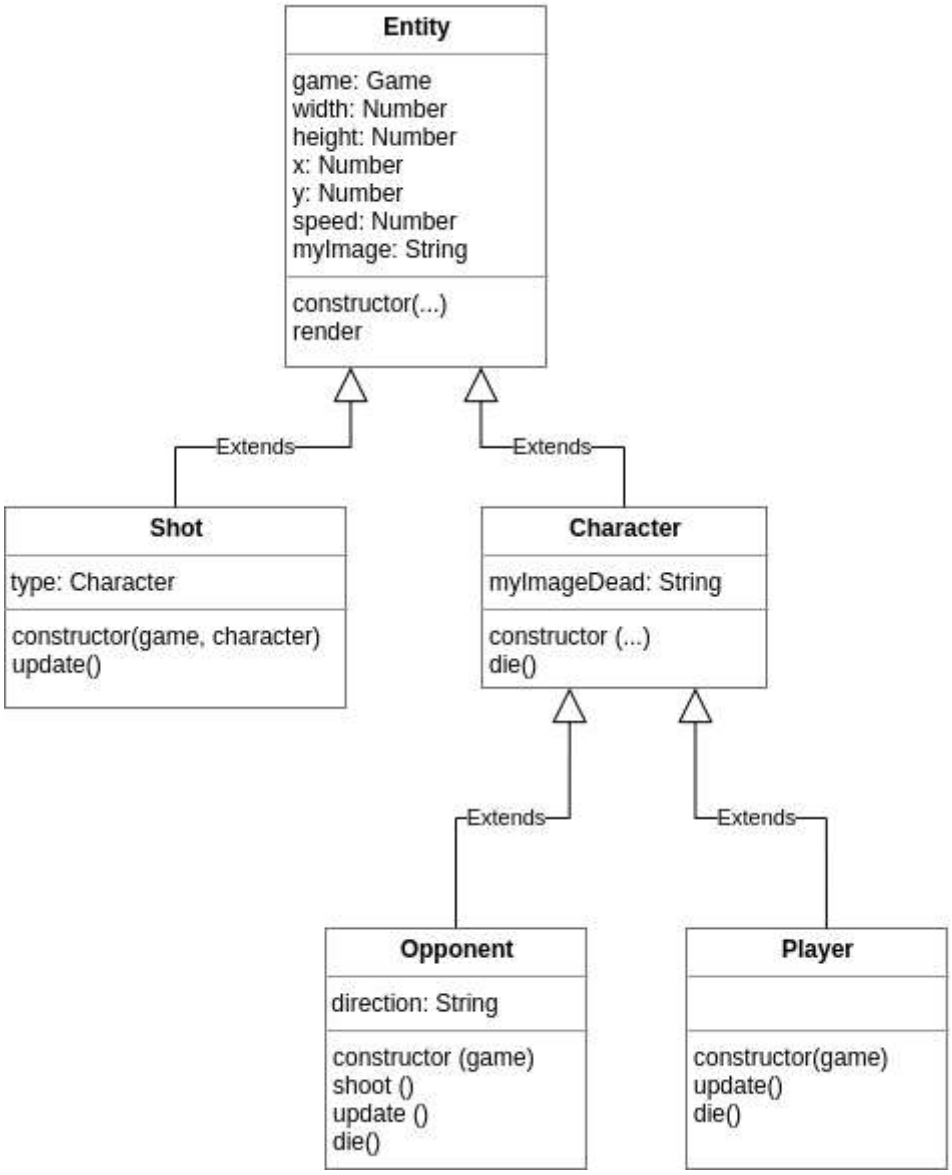
```
$ cd Entrega3_juego
```

Elementos del juego

En este código, para modelar cada uno de los elementos del juego empleamos una clase JavaScript con sus métodos y atributos, los cuales se describen a continuación:

- **Entity:** Cada uno de los elementos que se pintan en el juego
- **Character**: Cada uno de los personajes del juego, es decir, aquellos elementos que tienen "vida". Hereda de la clase *Entity*
- **Player**: Personaje principal del juego. Hereda de la clase *Character*
- **Opponent**: Forma a la que tenemos que convertir en estrella. Hereda de la clase *Character*
- **Shot**: Disparo de un *Character*. Hereda de la clase *Entity*
- **Game**: El propio juego

En el propio código están documentados todos los atributos y métodos de estas clases con detalle. El siguiente diagrama muestra la jerarquía de herencia de clases:



Comienzo y actualización del juego

En el fichero `index.html` se importan todos los scripts necesarios para el funcionamiento del juego, entre los que figuran todas las clases necesarias y el fichero `main.js`. En este fichero se definen una serie de constantes necesarias para el juego, se crea una instancia de la clase *Game* y se llama a su método *start* para comenzar la partida.

El método *start* crea los personajes, pinta el juego según el tamaño de la pantalla e inicializa los escuchadores de eventos (los cuales veremos en el siguiente apartado). Adicionalmente, en este método se da comienzo a un temporizador que llama a la función *update* cada 50 ms para actualizar y pintar el estado del juego actualizado según las acciones del usuario, de los movimientos del oponente y de la posición de los disparos. Este intervalo de tiempo es equivalente a 20 marcos por segundo, es decir, estamos cambiando lo que muestra el juego 20 veces cada segundo, más que suficiente para crear la ilusión de movimiento.

Manejo de eventos

Para poder manejar el personaje principal del juego con las flechas del teclado o con la pantalla táctil debemos hacer uso de los eventos que nos proporciona el navegador para este propósito. En el método *start* de la clase *Game*, inicializamos los escuchadores de eventos necesarios:

- **keydown** : Se llama cuando el usuario pulsa una tecla. Guarda la tecla pulsada en el atributo *keyPressed* de *Game*.
- **keyup** : Se llama cuando el usuario deja de pulsar una tecla. Elimina el contenido del atributo *keyPressed* de *Game*.
- **touchstart** : Se llama cuando el usuario toca la pantalla. Guarda la posición horizontal (x) donde el usuario ha tocado en el atributo *xDown* de *Game*.
- **touchmove** : Se llama cuando el usuario arrastra el dedo por la pantalla. Elimina el contenido del atributo *xDown* de *Game*.

Como hemos visto antes, cada 50ms se llama al método *update* de *Game*. Este método comprueba el valor de *xDown* y *keyPressed* para actualizar la posición del personaje principal en función de las acciones del usuario.

Tareas

Se pide modificar el código proporcionado para lograr tres funcionalidades nuevas:

- Registro de los **puntos conseguidos** por el usuario. Cada vez que convierta a un oponente en estrella debe incrementar el número de puntos en una unidad.
- El personaje principal debe contar con **tres vidas**. Si es alcanzado por un disparo, en vez de perder, el número de vidas disminuirá en una unidad, otorgándole una nueva oportunidad para ganar. Si el número de vidas llega a cero, se termina el juego.
- Si el jugador consigue disparar al oponente (triángulo) y convertirlo en estrella, se le presentará una **oponente final** más poderoso (pentágono). Éste se moverá al **doble de velocidad** que el triángulo.

Para implementar las tres funcionalidades debes seguir los siguientes pasos:

1. Añadir un atributo nuevo *score* a la clase *Game* que refleje la puntuación (inicialmente 0).
2. Modificar el código del método *die* de la clase *Opponent* para que sume un punto a *score* cada vez que se dispara a un triángulo.
3. Añadir un atributo nuevo *lives* a la clase *Player* que valga 3 inicialmente. Puedes definir el nº de vidas inicial en una constante en main.js.
4. Modificar el código del método *die* de la clase *Player* para que reste una vida cada vez que al jugador le alcance un disparo. Sólo debe morir si el nº de vidas es cero tras la resta.
5. Añadir el código necesario para pintar la puntuación y las vidas en la pantalla del juego en todo momento. Para ello crea una lista (etiqueta ul de HTML) con dos elementos (etiqueta li). El primero, con id "scoreli", mostrará la puntuación con el siguiente formato: `Score: x` , siendo `x`

el valor del atributo *score* del juego. El segundo, con id `livesli`, mostrará el nº de vidas con el siguiente formato: `Lives: y`, siendo `y` el valor del atributo *lives* del jugador.

6. Crear una clase nueva llamada *Boss* en un nuevo fichero llamado Boss.js (no te olvides de importarlo en game.html). Esta clase debe heredar los métodos y atributos necesarios de la clase *Opponent* sobreescribiendo aquellos que sean necesarios para lograr la funcionalidad requerida. Para representar al jefe final puedes usar las imágenes `jefe.png` y `jefe_muerto.png` de la carpeta assets.
7. Modificar el código necesario para que cuando el jugador consiga matar al triángulo, le aparezca el desafío final. Es decir, el atributo opponent de la instancia de `Game` debe contener un objeto `Boss` cuando el jugador derrote al oponente inicial.
8. Modificar el código de la función `endGame` (no modificar la cabecera) para que, si el jugador consigue derrotar al jefe final gane la partida aparezca la imagen `you_win.png` de la carpeta assets, en vez de `game_over.png`.

Prueba de la práctica

Para ayudar al desarrollo, se provee una herramienta de autocorrección que prueba las distintas funcionalidades que se piden en el enunciado. Para utilizar esta herramienta debes tener node.js (y npm) (https://nodejs.org/es/) y Git instalados.

Para instalar y hacer uso de la herramienta de autocorrección en el ordenador local, ejecuta los siguientes comandos en el directorio del proyecto:

```
$ npm install -g autocorector     ## Instala el programa de test
$ autocorector                    ## Pasa los tests al fichero a entregar
.........................         ## en el directorio de trabajo
... (resultado de los tests)
```

También se puede instalar como paquete local, en el caso de que no se dispongas de permisos en el ordenador desde el que estás trabajando:

```
$ npm install autocorector     ## Instala el programa de test
$ npx autocorector             ## Pasa los tests al fichero a entregar
.........................      ## en el directorio de trabajo
... (resultado de los tests)
```

Se puede pasar la herramienta de autoorrección tantas veces como se desee sin ninguna repercusión en la calificación.

Instrucciones para la Entrega y Evaluación.

Una vez satisfecho con su calificación, el alumno puede subir su entrega a Moodle con el siguiente comando:

```
$ autocorector --upload
```

o, si se ha instalado como paquete local:

```
$ npx autocorector --upload
```

La herramienta de autocorrección preguntará por el correo del alumno y el token de Moodle. En el enlace https://www.npmjs.com/package/autocorector se proveen instrucciones para encontrar dicho token.

RÚBRICA: Se puntuará el ejercicio a corregir sumando el % indicado a la nota total si la parte indicada es correcta:

- **25%:** Muestra correctamente las vidas del usuario
- **25%:** Muestra correctamente la puntuación del usuario
- **50%:** La funcionalidad del oponente final está implementada correctamente

Si pasa todos los tests se dará la máxima puntuación.

Entrega 3 - Juego de disparos

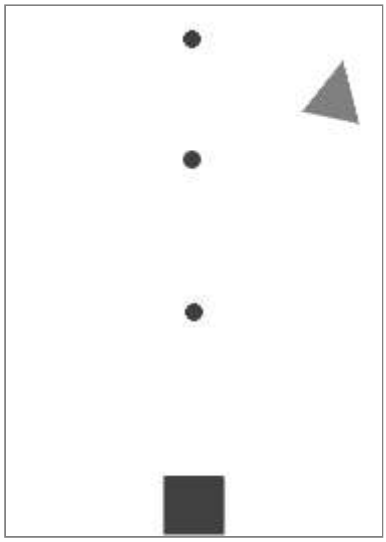
Versión: 18 de Febrero de 2020

Objetivo

Practicar con clases, Booleans, Strings y con el manejo de eventos.

Descripción de la práctica

En esta entrega vamos a desarrollar un juego completo usando HTML, CSS y JavaScript. El juego consiste en un juego clásico de disparos, en el que manejaremos a nuestro personaje (cuadrado) utilizando las flechas del teclado o la pantalla táctil. El objetivo del juego es disparar a una serie de formas que aparecerán en la pantalla para convertirlas en estrellas, a la vez que esquivamos sus disparos. Para comenzar el desarrollo partimos de la versión básica del juego cuyo código proporcionamos.



Descargar el código del proyecto

El proyecto se descarga en el ordenador local con estos comandos:

El proyecto debe clonarse en el ordenador desde el que se está trabajando

```
$ git clone https://github.com/CORE-2020/Entrega3_juego
```

Entrar en el directorio de trabajo

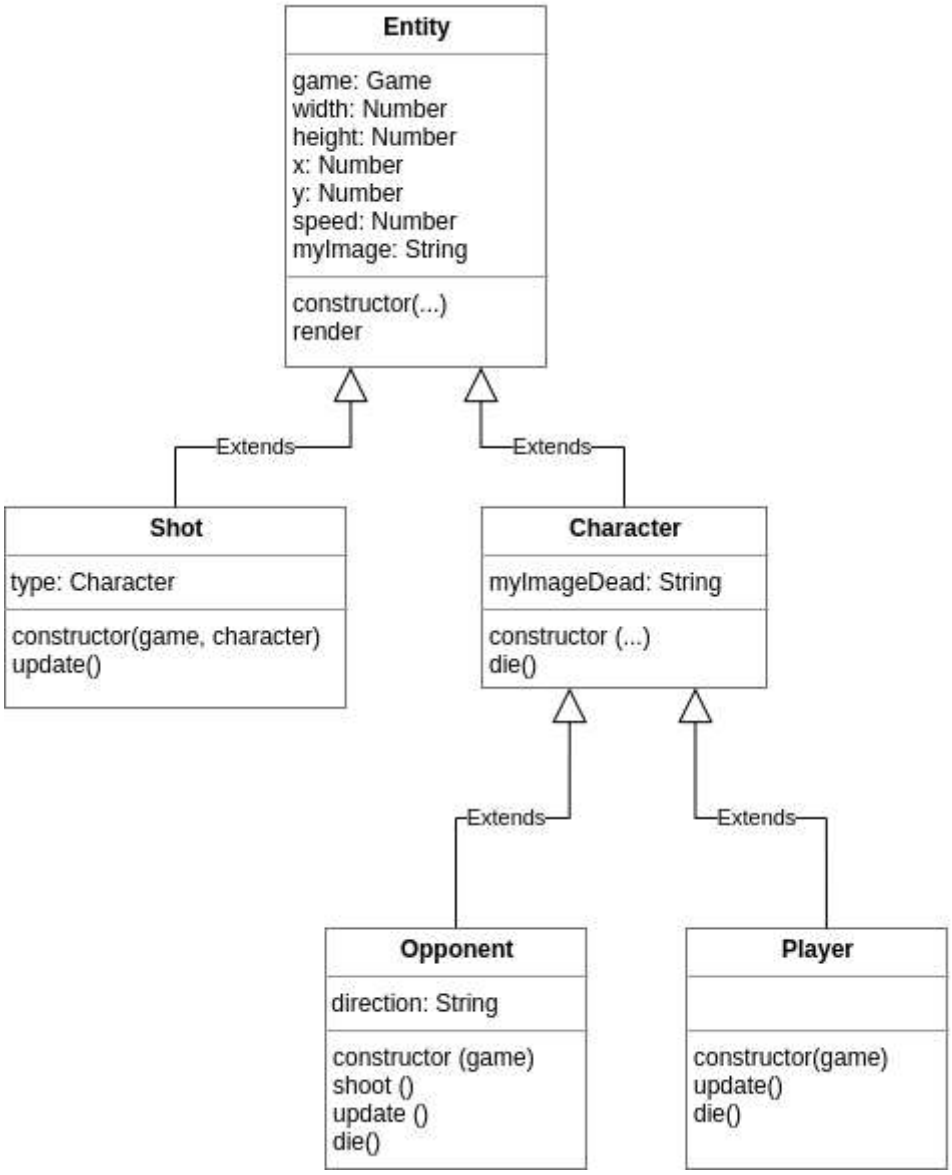
```
$ cd Entrega3_juego
```

Elementos del juego

En este código, para modelar cada uno de los elementos del juego empleamos una clase JavaScript con sus métodos y atributos, los cuales se describen a continuación:

- **Entity:** Cada uno de los elementos que se pintan en el juego
- **Character**: Cada uno de los personajes del juego, es decir, aquellos elementos que tienen "vida". Hereda de la clase *Entity*
- **Player**: Personaje principal del juego. Hereda de la clase *Character*
- **Opponent**: Forma a la que tenemos que convertir en estrella. Hereda de la clase *Character*
- **Shot**: Disparo de un *Character*. Hereda de la clase *Entity*
- **Game**: El propio juego

En el propio código están documentados todos los atributos y métodos de estas clases con detalle. El siguiente diagrama muestra la jerarquía de herencia de clases:



Comienzo y actualización del juego

En el fichero `index.html` se importan todos los scripts necesarios para el funcionamiento del juego, entre los que figuran todas las clases necesarias y el fichero `main.js`. En este fichero se definen una serie de constantes necesarias para el juego, se crea una instancia de la clase *Game* y se llama a su método *start* para comenzar la partida.

El método *start* crea los personajes, pinta el juego según el tamaño de la pantalla e inicializa los escuchadores de eventos (los cuales veremos en el siguiente apartado). Adicionalmente, en este método se da comienzo a un temporizador que llama a la función *update* cada 50 ms para actualizar y pintar el estado del juego actualizado según las acciones del usuario, de los movimientos del oponente y de la posición de los disparos. Este intervalo de tiempo es equivalente a 20 marcos por segundo, es decir, estamos cambiando lo que muestra el juego 20 veces cada segundo, más que suficiente para crear la ilusión de movimiento.

Manejo de eventos

Para poder manejar el personaje principal del juego con las flechas del teclado o con la pantalla táctil debemos hacer uso de los eventos que nos proporciona el navegador para este propósito. En el método *start* de la clase *Game*, inicializamos los escuchadores de eventos necesarios:

- **keydown** : Se llama cuando el usuario pulsa una tecla. Guarda la tecla pulsada en el atributo *keyPressed* de *Game*.
- **keyup** : Se llama cuando el usuario deja de pulsar una tecla. Elimina el contenido del atributo *keyPressed* de *Game*.
- **touchstart** : Se llama cuando el usuario toca la pantalla. Guarda la posición horizontal (x) donde el usuario ha tocado en el atributo *xDown* de *Game*.
- **touchmove** : Se llama cuando el usuario arrastra el dedo por la pantalla. Elimina el contenido del atributo *xDown* de *Game*.

Como hemos visto antes, cada 50ms se llama al método *update* de *Game*. Este método comprueba el valor de *xDown* y *keyPressed* para actualizar la posición del personaje principal en función de las acciones del usuario.

Tareas

Se pide modificar el código proporcionado para lograr tres funcionalidades nuevas:

- Registro de los **puntos conseguidos** por el usuario. Cada vez que convierta a un oponente en estrella debe incrementar el número de puntos en una unidad.
- El personaje principal debe contar con **tres vidas**. Si es alcanzado por un disparo, en vez de perder, el número de vidas disminuirá en una unidad, otorgándole una nueva oportunidad para ganar. Si el número de vidas llega a cero, se termina el juego.
- Si el jugador consigue disparar al oponente (triángulo) y convertirlo en estrella, se le presentará una **oponente final** más poderoso (pentágono). Éste se moverá al **doble de velocidad** que el triángulo.

Para implementar las tres funcionalidades debes seguir los siguientes pasos:

1. Añadir un atributo nuevo *score* a la clase *Game* que refleje la puntuación (inicialmente 0).
2. Modificar el código del método *die* de la clase *Opponent* para que sume un punto a *score* cada vez que se dispara a un triángulo.
3. Añadir un atributo nuevo *lives* a la clase *Player* que valga 3 inicialmente. Puedes definir el nº de vidas inicial en una constante en main.js.
4. Modificar el código del método *die* de la clase *Player* para que reste una vida cada vez que al jugador le alcance un disparo. Sólo debe morir si el nº de vidas es cero tras la resta.
5. Añadir el código necesario para pintar la puntuación y las vidas en la pantalla del juego en todo momento. Para ello crea una lista (etiqueta ul de HTML) con dos elementos (etiqueta li). El primero, con id "scoreli", mostrará la puntuación con el siguiente formato: `Score: x` , siendo `x`

el valor del atributo *score* del juego. El segundo, con id `livesli`, mostrará el nº de vidas con el siguiente formato: `Lives: y`, siendo `y` el valor del atributo *lives* del jugador.

6. Crear una clase nueva llamada *Boss* en un nuevo fichero llamado Boss.js (no te olvides de importarlo en game.html). Esta clase debe heredar los métodos y atributos necesarios de la clase *Opponent* sobreescribiendo aquellos que sean necesarios para lograr la funcionalidad requerida. Para representar al jefe final puedes usar las imágenes `jefe.png` y `jefe_muerto.png` de la carpeta assets.
7. Modificar el código necesario para que cuando el jugador consiga matar al triángulo, le aparezca el desafío final. Es decir, el atributo opponent de la instancia de `Game` debe contener un objeto `Boss` cuando el jugador derrote al oponente inicial.
8. Modificar el código de la función `endGame` (no modificar la cabecera) para que, si el jugador consigue derrotar al jefe final gane la partida aparezca la imagen `you_win.png` de la carpeta assets, en vez de `game_over.png`.

Prueba de la práctica

Para ayudar al desarrollo, se provee una herramienta de autocorrección que prueba las distintas funcionalidades que se piden en el enunciado. Para utilizar esta herramienta debes tener node.js (y npm) (https://nodejs.org/es/) y Git instalados.

Para instalar y hacer uso de la herramienta de autocorrección en el ordenador local, ejecuta los siguientes comandos en el directorio del proyecto:

```
$ npm install -g autocorector     ## Instala el programa de test
$ autocorector                    ## Pasa los tests al fichero a entregar
.........................         ## en el directorio de trabajo
... (resultado de los tests)
```

También se puede instalar como paquete local, en el caso de que no se dispongas de permisos en el ordenador desde el que estás trabajando:

```
$ npm install autocorector     ## Instala el programa de test
$ npx autocorector             ## Pasa los tests al fichero a entregar
.........................      ## en el directorio de trabajo
... (resultado de los tests)
```

Se puede pasar la herramienta de autoorrección tantas veces como se desee sin ninguna repercusión en la calificación.

Instrucciones para la Entrega y Evaluación.

Una vez satisfecho con su calificación, el alumno puede subir su entrega a Moodle con el siguiente comando:

```
$ autocorector --upload
```

o, si se ha instalado como paquete local:

```
$ npx autocorector --upload
```

La herramienta de autocorrección preguntará por el correo del alumno y el token de Moodle. En el enlace https://www.npmjs.com/package/autocorector se proveen instrucciones para encontrar dicho token.

RÚBRICA: Se puntuará el ejercicio a corregir sumando el % indicado a la nota total si la parte indicada es correcta:

- **25%:** Muestra correctamente las vidas del usuario
- **25%:** Muestra correctamente la puntuación del usuario
- **50%:** La funcionalidad del oponente final está implementada correctamente

Si pasa todos los tests se dará la máxima puntuación.