MONTAÑISMO v1.0

Realizado por:
Manuel Esteban Carrillo Calderón
Andrea Díez Barroso

Fecha: 11/01/2017

Contenido

1 Descripción general del proyecto

1.1 Introducción

El proyecto a realizar consiste en la creación de un videojuego en JavaScript que se ejecutará en un servidor web. El videojuego será simple, sin mucha complicación en los gráficos y la jugabilidad, aunque por su propia naturaleza sí en la parte de física.

En los siguientes apartados se va a describir de forma más detallada las funcionalidades del juego, el género al que pertenece, las tácticas a emplear, los recursos a utilizar, entre otros.

1.2 Descripción general del videojuego

Después de haber discutido varias opciones sobre la temática del videojuego al final nos hemos decantado por uno que, aunque se asemeje un poco, no es el típico plataformas, sino que se trata de una aventura en la que el jugador se mete en el papel de un conductor que debe dirigir un vehículo por un terreno que se va complicando con cada nivel.

Por la propia naturaleza del juego, éste va a ser 2D en cuyo mapa principal se podrá observar uno de los tres vehículos disponibles (automóvil, todoterreno y coche monstruo), el camino a recorrer, los obstáculos, así como los controles y varios elementos que indiquen al usuario el estado del mismo.

1.3 Funcionalidad

Las reglas del juego son sencillas, avanzar por un terreno que puede ser llano, con cuestas, abismos, algún obstáculo y solo puede utilizar el acelerador y el freno.

El objetivo de esta elección es conseguir llegar a la meta de cada mapa, sabiendo que cada nivel tendrá un mapa diferente y unas características especiales, como puede ser un cambio en la gravedad (por ejemplo, si se conduce el vehículo en la superficie lunar) o en la fricción del vehículo con el suelo (por ejemplo, un suelo cubierto de barro). Además, se tendrá que llegar a determinados objetivos en cada nivel, por ejemplo, cada X metros recorridos en el terreno se tendrá que recoger combustible, que se podría ver como la *stamina* del personaje principal.

A continuación, vamos a mostrar un esquema general de los elementos con los que actualmente cuenta el juego:

Vehículos

- 4x4: coche todoterreno conducido por un chico
- Automóvil: vehículo de baja potencia conducido por una chica.
- Coche Monstruo: vehículo pesado y potente conducido por un orco.

Mapas

- Campo: ruta sencilla e ideal para iniciar la aventura.
- Cueva: trayecto donde no podremos ver el cielo.
- Luna: apenas hay gravedad.

Mejoras en vehículos

- Motor: actualización para los motores de los vehículos, los hace más potentes y más rápidos.
- Frenos: mejora general en el sistema de frenado.
- Transmisión: mejora tracción en las cuatro ruedas, más potencia y mejor manejo.
- Suspensión: el vehículo podrá conducirse mejor en los baches.

Elementos de juego

- Monedas: cada mapa cuenta con una cierta cantidad de monedas que el jugador puede coger.
- Combustible: Disponible en los mapas para X distancia.

El usuario podrá elegir el vehículo que desee para cada nivel, teniendo en cuenta que cada uno tiene una serie de características propias, como pueden ser la potencia, la velocidad máxima, el rendimiento, etc. De la misma manera puede elegir el mapa en que va a jugar: al igual que sucede con los coches cada uno tiene una serie de características distintas que hacen que la conducción varíe y que el usuario tenga que adaptarse a las condiciones para poder llegar a la meta.

El jugador tiene la posibilidad de hacer *upgrades* sobre las características del vehículo, esto es, mejorar su rendimiento. Para poder realizar estas mejoras sobre algún coche se utiliza la puntuación obtenida en cada nivel por recolectar monedas, esta puntuación se va sumando al dinero del jugador. Antes de empezar una partida, se muestra una pantalla con las mejoras descritas anteriormente y el jugador puede elegir en cuál de ellas invertir su dinero. Al llevar a cabo una mejora, el precio de la misma aumenta para la siguiente fase.

Cada nivel puede finalizar de dos formas: la manera exitosa es que el vehículo consiga llegar al final del terreno, en cuyo caso puede continuar al siguiente nivel, o que el vehículo en el trayecto sufra algún inconveniente como el agotamiento del combustible o que se encuentre en una posición que le sea imposible de continuar (p.ej. la cabeza del conductor ha chocado con el terreno provocándole la ¡muerte!).

En esta entrega final hemos cumplido con todos los objetivos propuestos al inicio, aunque en el juego aún quedan muchas cosas por hacer y a su vez mejorar debido al poco tiempo que tenemos para su desarrollo.

1.4 Dominio, o género, de desarrollo del proyecto.

Al contar con elementos de juego como los vehículos, los caminos, los controles del acelerador y el freno, entre otros, este proyecto podríamos clasificarlo en un juego de "conducción".

1.5 Entorno y herramientas de desarrollo.

Para el desarrollo del proyecto utilizamos el motor de juego *Phaser* junto al lenguaje de programación *javascript* y como entorno de trabajo usamos *Brackets*, un editor de texto focalizado en el desarrollo web que ofrece funcionalidades muy útiles a la hora de programar y probar el código del juego, por ejemplo, nos permite correr un servidor desde el mismo editor de texto.

Para la creación de niveles utilizamos el programa *Tiled Map Editor*, muy útil ya que nos permite crear una infinidad de mapas a nuestro gusto y conveniencia. También utilizamos el programa *Paint.NET* para crear y hacer pequeños ajustes en los sprites del juego y la aplicación *Garage Band* para el crear el track musical.

1.6 Descripción del hardware.

Al ser un videojuego sin mucha complejidad en sus controles, los dispositivos hardware que se necesitarán son: un teclado desde el que se van a controlar los movimientos del vehículo (acelerar y frenar), un ratón para facilitar la selección de niveles, vehículos y realizar configuración en el juego, y preferentemente (aunque no indispensable) un dispositivo de salida de audio para escuchar por ejemplo el ruido del motor de un coche y también, de fondo, un track musical para entretener al jugador.

1.7 Equipo y lugar de trabajo.

El proyecto se va ha desarrollado entre dos personas haciendo uso de los elementos hardware y software mencionados en los apartados anteriores en el horario de prácticas de la asignatura, así como fuera de él, ya que es muy complicado crear un videojuego de esta magnitud únicamente en horas de clase.

1.8 Técnicas a emplear.

En la realización de este proyecto utilizamos una metodología de desarrollo incremental, realizando una fase de análisis de requisitos, de diseño, una de implementación, una fase de pruebas de lo desarrollado y una fase de integración, con su constante mantenimiento. De esta manera controlamos la complejidad del producto y los posibles riesgos a futuro.

Inicialmente también realizamos una estimación del tiempo que una tarea llevaría, y una planificación temporal para su fase de inicio y elaboración. Según avanzamos en el videojuego intentamos cumplir los plazos estimados para la elaboración de cada tarea y de esta forma llevamos un control sobre lo hecho, lo realizado y lo restante.

Esta forma de llevar el proyecto ha evitado posibles retrasos y conflictos en las versiones de desarrollo del videojuego.

1.9 Recursos adicionales

En primera instancia, nuestro proyecto no necesita de recursos adicionales a los ya mencionados ya que la parte gráfica no es muy complicada, basta con incluir unos *sprites* para los vehículos y posibles obstáculos, y diversos fondos para los escenarios. En lo referente al sonido se incluimos un track musical y sonidos característicos del juego, por ejemplo, el sonido del motor, de las colisiones, etc.

Uno de los recursos usados para la implementación del escenario han sido las polilineas, unos objetos con los que podemos crear líneas de colisión curvas y así hacer más real la subida y bajada por colinas.

El track musical del juego lo creamos nosotros mediante la aplicación Garage Band, solo se encuentra disponible para dispositivos de Apple.

1.10 Videojuegos existentes en los que basamos nuestro proyecto.

Varias de las ideas tomadas para nuestro proyecto tienen como origen un videojuego ya existente.

Este videojuego se llama **Hill Climbing**, está disponible para dispositivos móviles, también hay algunas versiones en Web.

Figura 1.1

Como se puede apreciar en la imagen 1.1, se trata de un vehículo, y sus correspondientes mandos. Además de los objetos como monedas o garrafas de gasolina. También se ve la distancia recorrida, así como la cantidad de monedas recogidas y el depósito de gasolina.

Figura 1.2

El mapa tiene diferentes elementos además de colinas, como puentes que tiemblan al pasar por ellos.

Figura 1.3

En esta última captura, mostramos qué sucede si el vehículo vuelca, evidentemente el nivel acaba y el jugador ha perdido la partida.

Para la realización del videojuego hemos decidido basarnos en un ejemplo de phaser que nos ofrece una versión muy simple para el videojuego, pero en la cual nos basaremos para ir construyendo poco a poco nuestro proyecto respecto a lo descrito anteriormente:

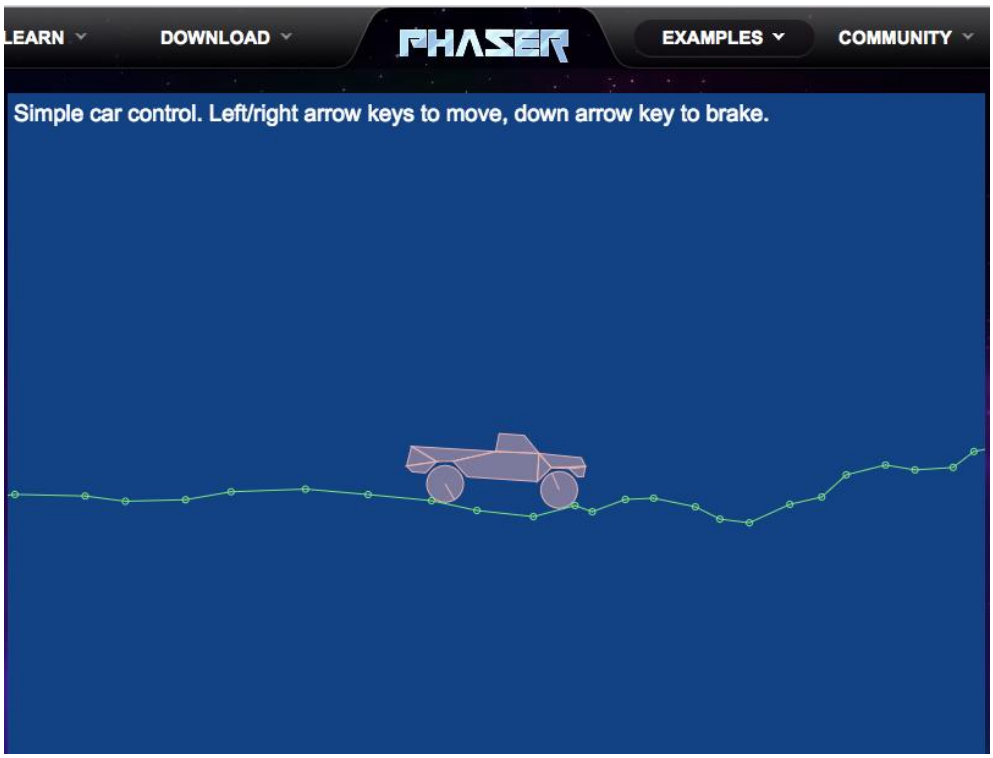


Figura 1.4

Aquí está el enlace para probar el ejemplo escogido: Car on Terrain

Este es el enlace a una web de juegos con el más similar al de las capturas: Hill Climbing

2 Análisis/Diseño

2.1 Gestión de directorios

A continuación, mostramos la forma en la que se encuentran organizados los archivos en sus correspondientes directorios y comentamos los más importantes:

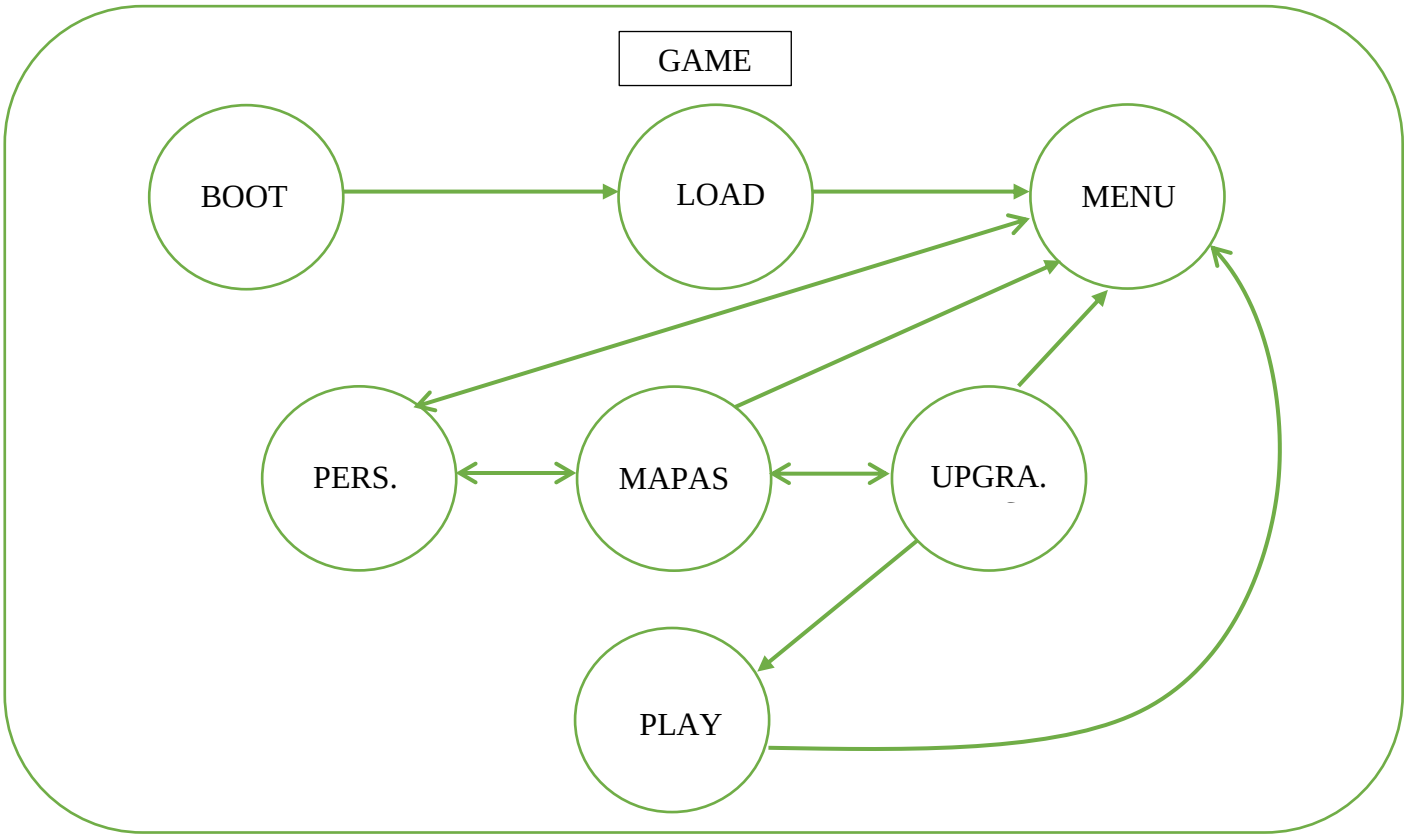
Assets: Aquí se encuentran los elementos audio-visuales utilizados.
Modules: Los archivos contenidos en este directorio representan las distintas clases con las que cuenta el juego. Se han separado en subdirectorios para agruparlas las de funcionalidad similar.
States: En este directorio están los estados del juego que son los que se encargan de mostrar las distintas pantallas.

2.2 Estados

Para la creación de las distintas pantallas del videojuego de forma organizada hemos creado varios estados ya que cada uno de ellos mantiene sus propias variables y objetos, por lo que no es posible modificar objetos de un estado en otros estados, con las excepciones de los assets y las variables globales.

Los estados con los que contamos en esta versión final son: Boot, Load, Menu, Personajes, Mapas, Upgrades, Play, cada uno se corresponde con un archivo .js con el mismo nombre:

- Boot.js: Carga los sprites y elementos que serán utilizados en la pantalla de carga e inicializa el juego con la física P2.
- Load.js: Carga los recursos del juego, y muestra los elementos de la pantalla de carga.
- Menu.js: Muestra la pantalla del menú y nos permite iniciar el juego.
- Personajes.js: Da la posibilidad de elegir uno de los tres personajes presentes en el juego.
- Mapas.js: Muestra al usuario los mapas existentes para su elección.
- Upgrades.js: En esta pantalla se muestran las mejoras posibles para el coche seleccionado y da la posibilidad de aplicarlas (comprarlas).
- Play.js: Se encarga de crear el mundo y permitir la jugabilidad.
- Game.js: este fichero inicializa Phaser, crea variables globales, carga los estados y arranca el primero (boot.js).

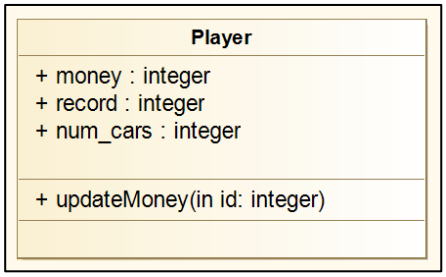


2.3 Clases

Para permitir la máxima funcionalidad del juego, hemos creado cinco clases, aunque en realidad son objetos JavaScript ya que es un lenguaje basado en prototipos que no contiene ninguna declaración de clase, como se encuentra, por ejemplo, en C# o Java. En su lugar, JavaScript utiliza funciones como clases. Definir una clase es tan fácil como definir una función.

2.3.1 Player

Esta clase contiene los atributos esenciales para el manejo de un usuario en el juego, como es la cantidad de dinero que tiene, la puntuación máxima alcanzada en las partidas, el conjunto de vehículos que posee, así como el número de ellos. El método *updateMoney* se encarga de incrementar o decrementar el dinero del jugador tras una partida o después de hacer una actualización.



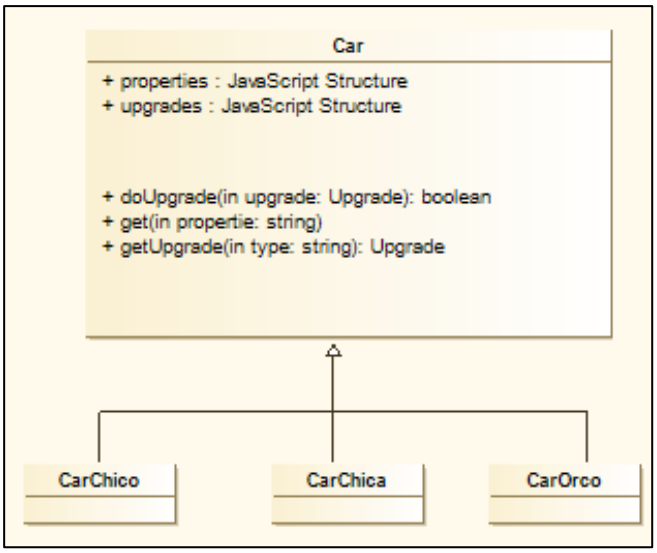
Clase con la que tiene relación: Car.

2.3.2 Car

Clase que se encarga del tratamiento de un coche y sus propiedades. De esta clase heredan otras tres que equivalen a los tres coches que tenemos, llamadas CarChico, CarChica y CarOrco.

La clase principal contiene dos atributos, el primero es una estructura JavaScript que representa las propiedades del coche (velocidad, aceleración, etc.) y el segundo otra estructura en la que se almacenan las mejoras que se pueden aplicar al coche.

Para llevar a cabo una actualización sobre el vehículo se cuenta con el método *doUpgrade.*

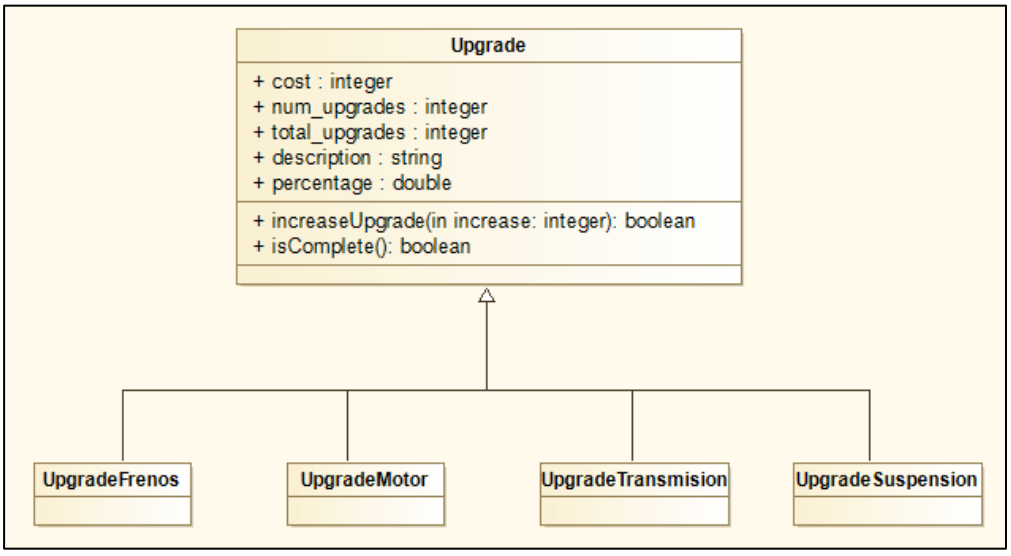


Clases con la que tiene relación: Player y Upgrade.

2.3.3 Upgrade

Esta clase contiene los atributos necesarios que describen una actualización general. Existen cuatro clases que heredan de ella que se corresponden con las distintas mejoras que se pueden aplicar al vehículo (motor, frenos, suspensión y transmisión).

Cada mejora tiene su propio coste, una descripción, un número total de actualizaciones que define el número de veces que se puede llevar a cabo (p.ej. la mejora del motor se puede realizar cuatro veces) y un porcentaje en el que se mejora. El método *increaseUpgrade* incrementa en uno la etapa de la actualización y el método *isComplete* comprueba si se ha llegado a su punto máximo.



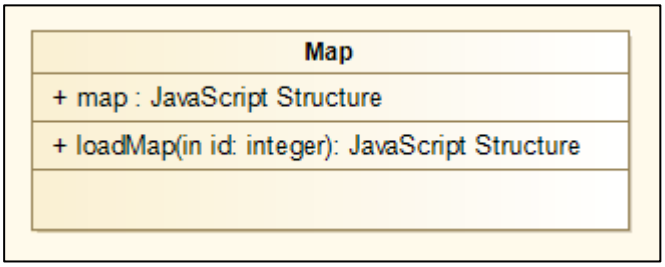
Clases con la que tiene relación: Car.

2.3.4 Map

Clase que se encarga de la creación del mapa en el que se va a jugar. Contiene una estructura que representa las propiedades del mapa, la cual se obtiene al llamar a la función *loadMap* con el identificador del mapa que se desee cargar. La estructura con de los mapas tiene la siguiente forma:

```
map = {
        id: entero identificador del mapa,
        name: nombre,
        tilesetImage: nombre del tileset cargado,
        tilemap: nombre del tilemap cargado,
        layer: nombre de la capa,
        gravity: gravedad del mundo,
        height: altura,
        width: anchura
};
```

Esta clase no tiene relación con ninguna de las anteriores debido a que no existe una dependencia funcional entre ella y el resto de clases del juego.

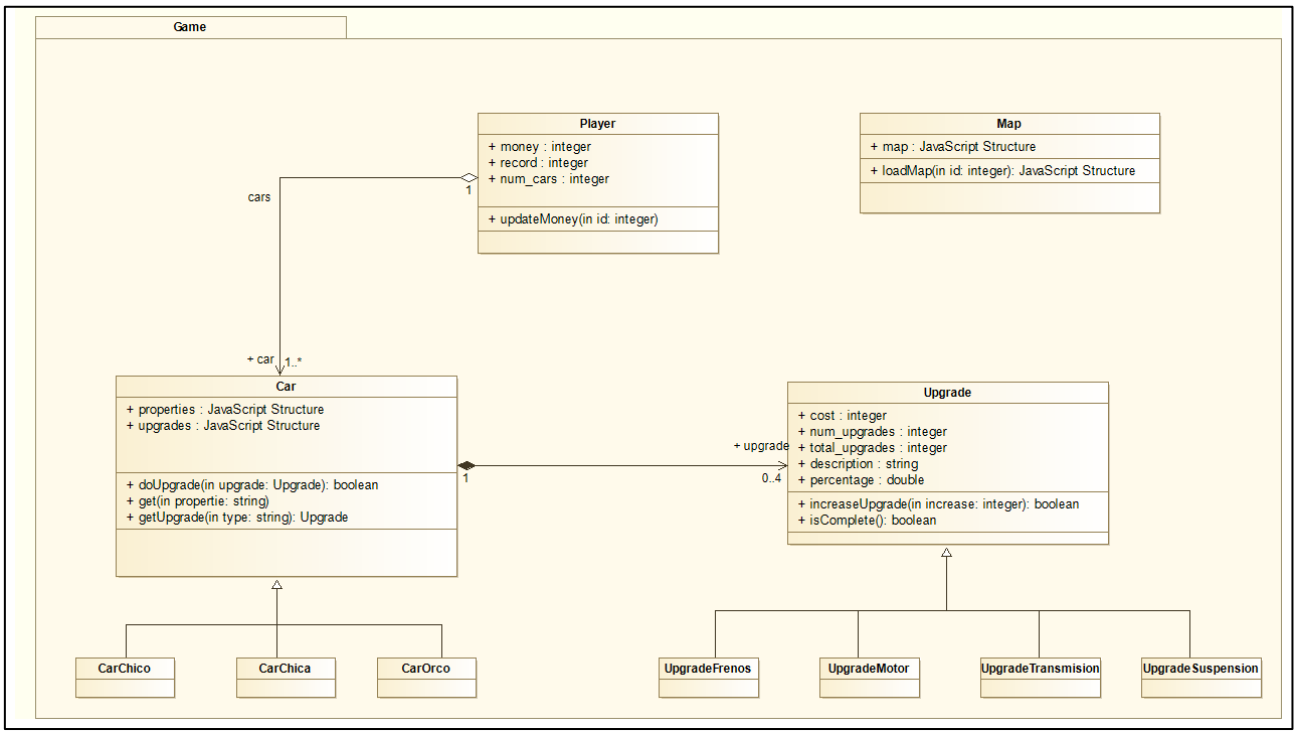


2.3.5 Message

Este módulo no contiene clases, pero si dos funciones (*UpgradeMessage* e *InformativeMessage)* que muestran en la pantalla un mensaje, el cual se puede tomar como un pop-up de aviso que se lanza al realizar una determinada acción. Estas funciones lo que hacen es cargar una imagen en la que se muestra una descripción de la acción que se va a realizar, son utilizadas básicamente para mostrar la información cuando se desea realizar una mejora o simplemente para mostrar un mensaje informativo al usuario.

2.3.6 Diagrama de Clases

Diagrama resultante de relacionar las clases descritas:



Hemos utilizado el patrón de diseño *Factory Method* para las clases Car y Upgrade ya que estamos definiendo una clase para crear un objeto, pero dejando en manos de las subclases la decisión de que clase concreta instaciar.

En el caso de los mapas, hemos aplicado el patrón *Abstract Factory* ya que definimos la clase para crear mapas pero sin especificar sus sub-clases concretas.

3 Implementación

Vamos a describir los elementos de implementación más destacados de cada fichero .js a continuación.

3.1 Desarrollo

3.1.1 Game.js:

En este fichero creamos la variable **game** que representa la entidad de nuestro juego, lo hacemos mediante una llamada a *new Phaser.Game()*. También creamos un espacio para almacenar las variables globales (de momento solo tenemos una para la puntuación), y añadimos a nuestra entidad de juego los distintos estados con los que contamos mediante. Después de esto llamamos al estado Boot.

3.1.2 Boot.js:

Contiene una función *preload* donde cargamos las imágenes que va a utilizar la pantalla de carga mediante la función de Phaser *game.load.image()* y *game.load.spritesheet()*.

En este fichero también se encuentra una función *create* donde inicializamos la física P2 *game.physics.startSystem(Phaser.Physics.P2JS)* y llamamos luego llamamos al estado Load.

3.1.3 Load.js:

Este fichero se gestiona la carga de los elementos necesarios para el juego, como son los sprites de los coches, de las monedas, del combustible, carga también los mapas y los sonidos del juego. Además, muestra un fondo de pantalla, el nombre del videojuego y una barra de progreso cuya animación se ajusta al tiempo de real de carga de los distintos assets, esto se consigue con la función de Phaser *this.load.setPreloadSprite(this.progressBar).*

Una vez cargado todo pasamos a mostrar el menú.

3.1.4 Menú.js:

Esta es la primera pantalla en la que el jugador puede interactuar ya que aparte de mostrar elementos como el fondo de pantalla, el título o la puntuación también cuenta con dos botones, uno para habilitar o deshabilitar el sonido y otro para mostrar el videojuego en pantalla completa.

Para dar funcionalidad al botón de sonido lo que hacemos utilizar una propiedad de la clase *Sound* de Phaser llamada *mute* la cual activa o desactiva el sonido al hacer click en el botón. La funcionalidad del botón de pantalla completa es similar al anterior, al hacer click sobre este botón llamamos al método *startFullScreen()* o *stopFullScreen()* de la clase *Scale* de Phaser si se desea entrar o salir del modo FullScreen, respectivamente.

Otra forma que el jugador tiene para interactuar con esta pantalla y las siguientes es presionar la tecla ENTER. Una vez hecho esto se pasa al siguiente estado Personajes.

3.1.5 Personajes.js:

Este fichero cumple la función de ser una pantalla de elección del personaje con el que se va a jugar. La elección está entre tres posibles personajes: una chica, un chico y un orco (No sabemos su género...). Por defecto en caso de que el usuario no elija ningún personaje clickando sobre cualquiera de ellos, el personaje que jugará será el chico.

Desde esta pantalla podemos volver a la pantalla de menú inicial, ir a la siguiente que se corresponde con la elección de mapas en el que se pueden jugar.

3.1.6 Mapas.js:

Este fichero forma la pantalla de elección de mapa en el que vamos a jugar. Para mostrar los mapas posibles usamos un carrusel. Para crear este carrusel hemos usado una librería que alguien creó y publicó en GitHub (Aquí). Para ello solo tenemos que hacer la carga de las imágenes, como para cualquier imagen del juego, crear un grupo de objetos para cada objeto del carrusel en el que se incluyan las imágenes a mostrar y un fondo. En nuestro caso usamos un fondo trasparente. Y creamos el carrusel tal y como el creador de la librería indica en su documentación y sus ejemplos.

Como describimos anteriormente tenemos tres posibles mapas en los que jugar, campo, cueva y luna. Cada uno tiene sus propiedades. Cuando uno de ellos está seleccionado aparece en un lado de la pantalla cuál de ellos ha sido elegido y por lo tanto en el mapa en el que se va a jugar.

3.1.7 Upgrades.js:

Se encarga de crear y mostrar una pantalla que permite llevar a cabo las distintas mejoras disponibles para el vehículo seleccionado. Al seleccionar cualquiera de las cuatro mejoras disponibles se abre un pop-up (utilizando las funciones de creación de mensajes descritas en el apartado 2.3) que muestra la descripción de la mejora, el coste de la misma y la etapa de mejora en la que se encuentra.

Para aplicar las mejoras se cuenta con el método *doUpgrade*, como éstas consumen el dinero del jugador, si no se cuenta con el suficiente crédito se muestra un mensaje informando del suceso. Lo mismo ocurre cuando la mejora ha llegado a su etapa final. Cada vez que se aplica alguna de las mejoras, su coste incrementa.

Desde esta pantalla se puede volver al inicio, a la pantalla anterior (selección de mapas) o si el jugador está listo, ¡empezar el juego!

3.1.8 Play.js:

Este es el fichero que más líneas de código tiene ya que es el encargado de proporcionar toda la jugabilidad respondiendo a las acciones del usuario. Cuenta con varias funciones, de las cuales vamos a comentar las más importantes:

- Create: Esta función se encarga de mostrar todos los elementos necesarios para el juego, como son los botones de sonido y FullScreen, la información sobre las monedas recogidas y el combustible del vehículo, así como añadir los distintos sonidos que tenemos. También dota de gravedad al mundo mediante *game.physics.p2.gravity.y* y hace una llamada a las funciones que crean el mundo (createWorld), el coche (createCar), las monedas (createCoins) y el combustible (createGas).

- CreateWorld: añade al juego los mapas creados mediante Tiled, permitiendo la colisión con las polilineas mediante la función *game.physics.p2.convertCollisionObjects().* Se da una información más detallada sobre el uso de esta función en el documento sobre polilineas adjunto.

- CreateCar: Esta función es la encargada crear conjunto del coche, que está divido en tres partes: las ruedas (delantera y trasera), el cuerpo (motor), y la cabeza del conductor. Estos elementos se crean a partir de los sprites disponibles y habilitamos la física P2 para ellos dándole una colisión en forma de rectángulo para el motor y de círculo para las ruedas y la cabeza del conductor.

  Para poder unir estos elementos de tal forma que su comportamiento se muestre lo más realista posible con respecto a un coche de verdad utilizamos principalmente dos funciones que nos proporciona el motor de física P2 de Phaser que son: createSpring() y createPrismaticConstraint(). Mediante la primera creamos algo parecido a un muelle entre objetos, que vendría a ser la suspensión entre las ruedas y el motor. Esta función nos permite dar propiedades como la rigidez del muelle, su longitud o la capacidad de 'botar' al tener contacto con el suelo. Como este 'muelle' no está fijo entre los objetos utilizamos la segunda función para añadir una restricción que entre ellos permitiendo así que las ruedas se mantengan limitadas superior e inferiormente con respecto al cuerpo del vehículo.

- CreateCoins y CreateGas: estas dos funciones se encargan de mostrar en la pantalla los sprites de las monedas y el combustible, cargando su posición desde una capa de objetos que contiene el mapa creado con Tiled. Una vez añadidos los sprites, se les proporciona física para permitir la colisión con el coche, la función *createBodyCallback()* nos permite hacer una llamada a una función específica cada vez que ocurra una colisión entre estos objetos, como por ejemplo llamar a takeCoin() o takeGas() para actualizar la puntuación o la cantidad de combustible.

- MoveCar: esta función se encarga de realizar los movimientos del coche, en nuestro caso de incrementar la velocidad (desde 0 hasta un máximo determinado) cuando se presionan las flechas izquierda o derecha y de frenar cuando se presiona la flecha hacia abajo. Cuando el coche se pone en movimiento, se activa el sonido del motor.

- GameOver: esta función es llamada cuando ocurre uno de los siguientes sucesos: el jugador muere porque la cabeza colisiona con el terreno, se

termina el combustible o se ha llegado al final del trayecto. En cualquier caso, muestra un mensaje de fin de juego y llama al estado Menu, donde se vuelve a repetir el proceso.

3.1.9 Index.html:

Este fichero lo que hace es cargar los ficheros .js e iniciar en un navegador todo el proceso de juego. Aquí hemos una imagen de fondo para la pantalla y hemos cargado una fuente de Google Fonts para mostrar la puntuación.

3.2 Manual de instrucciones

Por la forma en la que está desarrollado el videojuego, es muy intuitivo y fácil de jugar, los elementos que se necesitan son un teclado y un ratón. No hemos incluido soporte para gamepad debido a esta facilidad de utilización.

Para desplazarse por las distintas pantallas, del teclado se pueden utilizar las flechas o también la tecla ENTER. Haciendo uso del ratón también se puede desplazar por las distintas pantallas y a su vez seleccionar varios de los botones que tenemos habilitados, como el de pantalla completa, el de sonido o los de selección de personajes, mapas o mejoras.

Una vez que se empieza una partida se utilizan las flechas derecha, izquierda y hacia abajo para que el coche se mueva hacia adelante, hacia atrás o frene, respectivamente. Con el ratón se puede habilitar o deshabilitar el sonido y el modo de pantalla completa en cualquier momento.

A continuación, mostramos las distintas pantallas del juego y las acciones que se pueden realizar en cada una de ellas:

Menú inicial

1. **Botón de inicio:** Lleva al menú inicial (en este caso permanece en la misma pantalla).
2. **Etiqueta de dinero:** Muestra el presupuesto con el que cuenta el jugador.
3. **Botón de sonido:** Habilita o deshabilita los efectos de sonido presentes en el juego.
4. **Botón FullScreen:** Habilita o deshabilita el modo de pantalla completa.
5. **Título del juego.**
6. **Etiqueta de puntuación:** Muestra la puntuación alcanzada por el jugador al finalizar una partida.
7. **Etiqueta de record:** Muestra la máxima puntuación alcanzada de entre todas las partidas.
8. **Etiqueta informativa.**

Selección de personajes

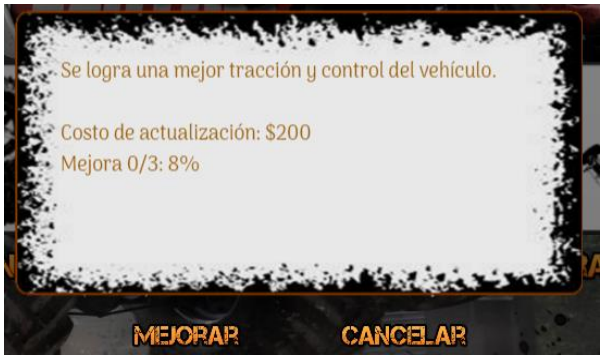
1. **Botón de inicio:** Lleva al menú inicial.
2. **Etiqueta de dinero:** Muestra el presupuesto con el que cuenta el jugador.
3. **Botón de sonido:** Habilita o deshabilita los efectos de sonido presentes en el juego.
4. **Botón FullScreen:** Habilita o deshabilita el modo de pantalla completa.
5. **Título de pantalla.**
6. **Personajes:** Muestra y pone a elección los personajes disponibles para el juego. Se seleccionan haciendo click sobre ellos. Si no se elige ninguno, por defecto se toma el chico o el personaje elegido en la última partida.
7. **Flecha derecha:** Al hacer click sobre ella pasa a la siguiente pantalla (Selección de niveles).

Selección de niveles

1. **Botón de inicio:** Lleva al menú inicial.
2. **Etiqueta de dinero:** Muestra el presupuesto con el que cuenta el jugador.
3. **Botón de sonido:** Habilita o deshabilita los efectos de sonido presentes en el juego.
4. **Botón FullScreen:** Habilita o deshabilita el modo de pantalla completa.
5. **Título de pantalla.**
6. **Mapas:** Muestra un carrusel con los mapas disponibles. Se seleccionan haciendo click sobre el botón 8. Si no se elige ninguno, por defecto se toma el mapa de campo o el elegido en la última partida.
7. **Selección:** Informa del mapa que ha sido seleccionado.
8. **Botón de selección:** Al hacer click sobre él, se selecciona el mapa que se muestra en ese momento en el carrusel.
9. **Flecha izquierda:** Lleva a la pantalla anterior (Selección de personajes.)
10. **Flecha derecha:** Lleva a la siguiente pantalla (Mejoras).

Mejoras

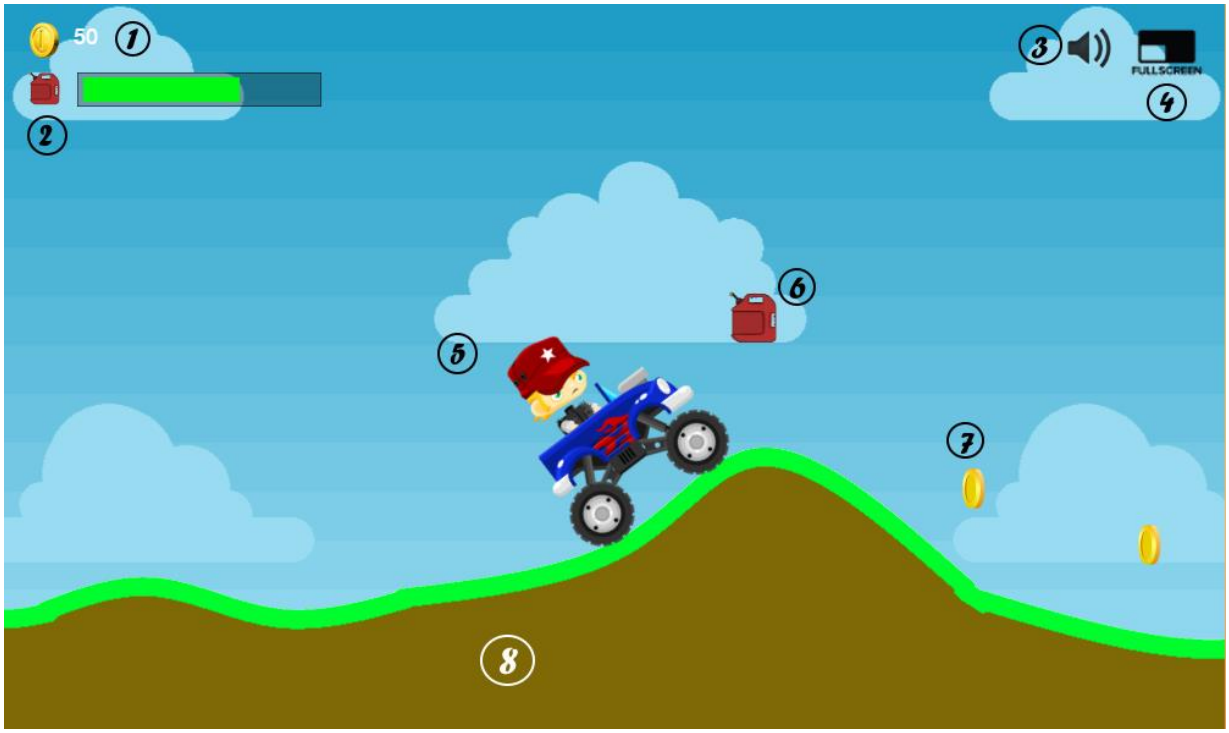
1. **Botón de inicio:** Lleva al menú inicial.
2. **Etiqueta de dinero:** Muestra el presupuesto con el que cuenta el jugador.
3. **Botón de sonido:** Habilita o deshabilita los efectos de sonido presentes en el juego.
4. **Botón FullScreen:** Habilita o deshabilita el modo de pantalla completa.
5. **Título de pantalla.**
6. **Mejoras:** Muestra los cuatro tipos de mejoras disponibles. Al hacer click sobre una de ellas se muestra un mensaje que muestra la descripción de la mejora y da la posibilidad de realizarla o de cancelar la operación. Ej.:



7. **Flecha izquierda:** Lleva a la pantalla anterior (Selección de personajes.)
8. **Flecha derecha:** Lleva a la siguiente pantalla (Mejoras).

Montañismo

Pantalla de Juego



1. **Dinero:** Muestra la cantidad de monedas que se van recogiendo en el mapa.
2. **Barra de combustible:** Muestra la cantidad de combustible actual del coche, se va reduciendo según avanza el juego y se llena cuando se recoge combustible. Si la barra llega a su fin, se acaba la partida.
3. **Botón de sonido:** Habilita o deshabilita los efectos de sonido presentes en el juego.
4. **Botón FullScreen:** Habilita o deshabilita el modo de pantalla completa.
5. **Vehículo.**
6. **Combustible:** Al recogerlo, llena la barra de combustible.
7. **Moneda:** Al recogerla, se aumenta el dinero.
8. **Terreno.**

A parte del contenido del videojuego, también incluimos una pequeña guía de cómo crear mapas en Tiled utilizando polilineas y cómo incluirlas en un juego creado con Phaser mediante un ejemplo.

4 Pruebas

Hemos realizado pruebas básicamente en la parte visual, por ejemplo, probamos que tipo de tweens se verían mejor en el juego, variamos los colores, variamos la colocación de los elementos en pantalla, etc. También comprobamos el comportamiento del coche cuando se colocan pendientes muy pronunciadas en el mapa e intentar subirlas antes y después de realizar una actualización sobre él, observando sus diferencias y ajustando el nivel de mejora.

En este apartado podríamos mencionar que tuvimos un bug extraño que hacía que al pasar por un sitio determinado del mapa el coche saltase automáticamente, revisamos cuidadosamente el mapa creado con Tiled y todo parecía correcto. Después de varias pruebas fallidas, movimos mínimamente un par de puntos de las polilineas en lugar que ocurría el salto y el error se solucionó.