DUNGEON'S ADVENTURE

Desarrollo de Aplicaciones Web
Modulo Proyecto C.F.G.S
DPTO.INFORMÁTICA – IES RIBERA DE CASTILLA

Autor: Alvaro Mínguez Alonso
Tutor: Jordi Pozo Cata
Curso: 2019/2020

ÍNDICE

1. INTRODUCCIÓN

El sector de los videojuegos en esta última década está en auge, es de las industrias que más recaudan en el año; pero hay un sector infravalorado: el sector de los videojuegos web. Dicho sector tiene las ventajas de no requerir un equipo potente para poder disfrutar de videojuegos, los cuales, en su mayoría, son gratis. Dada la situación que estamos viviendo actualmente, creo que es de especial importancia recalcar el impacto que tiene la tecnología en nuestras vidas. Por ello he querido crear un videojuego básico basado en el plataformeo clásico.

El objetivo es muy simple: recoger monedas con nuestro personaje para conseguir la mejor puntuación posible esquivando obstáculos. Tenemos 5 vidas por si fallamos.

MOTIVACIÓN

La motivación detrás de este proyecto proviene de mi pasión por los videojuegos. Desde pequeño he querido crear uno yo mismo. También, estando en la situación en la que estamos, creo que es el mejor momento para producir algo entretenido.

He utilizado Javascript puro, en vez de un framework (como podría ser *phaser*) porque quería aprender a utilizar programación orientada a objetos, puesto que en la empresa de prácticas utilizamos esta metodología, y esto me permitiría coger soltura con el lenguaje.

He utilizado también una arquitectura “mvc” (aunque sería más correcto referirse a ella como IPO; Input Processing, Output) puesto que me parece que así el código está mejor estructurado, los componentes son reutilizables y es más fácil de mantener.

2. RECURSOS UTILIZADOS

En este apartado voy a explicar el software y hardware necesarios para el desarrollo del proyecto.

2.1. JAVASCRIPT

He escogido Javascript, ya que es uno de los lenguajes más utilizados, y es soportado por la gran mayoría de navegadores actuales. Además su código puede ser ejecutado por casi cualquier equipo porque que es bastante ligero.

2.2. TILED

Tiled es un software que permite crear mapas de videojuegos y exportarlos como *csv*, *json,* etc. Su funcionamiento es simple; creas un mapa de X por Y, añades la imagen con los tiles o cuadrados con gráficos de la resolución que quieras y los emplazas en el mapa en blanco.

Su instalación y funcionamiento se explicarán en detalle más adelante.

2.3. VISUAL STUDIO CODE

Como editor de fuente he elegido Visual Studio Code puesto que la integración con git, la interfaz, snippets y plugins facilitan la tarea muchísimo.

2.4. GIT

He utilizado control de versiones, en específico Github para ir anotando los cambios en el código y mantener un histórico de las versiones del código.

2.5. PROGRAMACIÓN ORIENTADA A OBJETOS

He utilizado programación orientada a objetos por las ventajas que conlleva.

- **Reusabilidad**: Los objetos pueden ser reutilizados.
- **Mantenibilidad**: Es mucho más fácil de mantener, ya que es un código más simple de leer haciendo que abstraer errores sea menos complicado.
- **Escalabilidad**: Creando objetos nuevos se puede escalar el código más fácilmente.
- **Fiabilidad**: al aislar el código es mucho más sencillo encontrar errores y estos no se propagan al resto del código al estar aislados.

2.6. PISKEL

Para la parte gráfica del juego he utilizado una herramienta online llamada Piskel que se utiliza para crear pixelart. He utilizado también Adobe Photoshop y Paint para juntar todos los tiles en una misma imagen (esto suele ser llamado “tile set” en el mundo de los videojuegos) para hacer el dibujo del mapa, animaciones, etc.

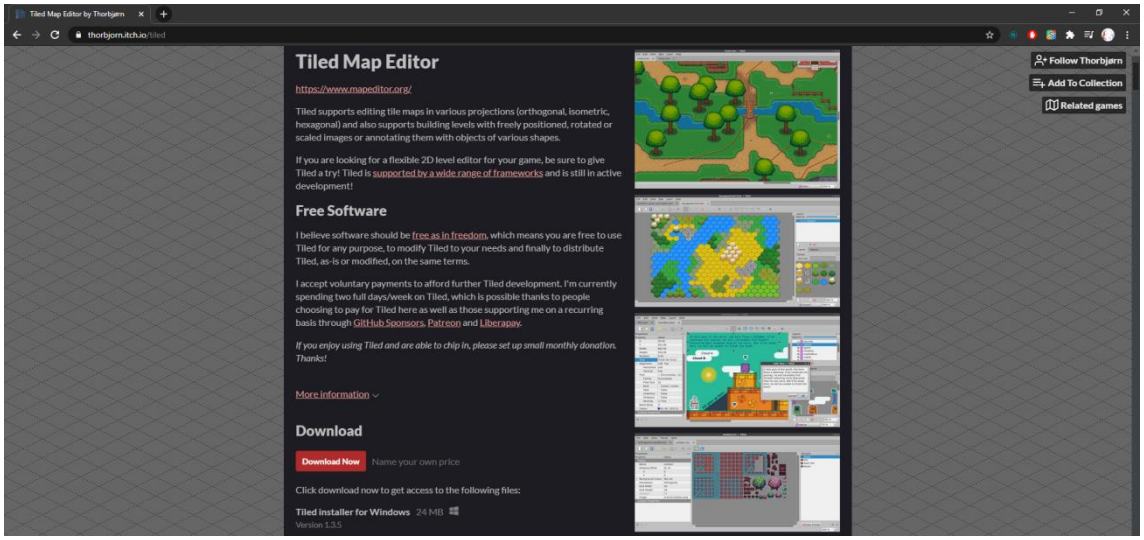
2.7. FIREBASE

He utilizado firebase para el despliegue de la aplicación debido a su sencillez de utilización.

3. EXPLICACIÓN DEL FUNCIONAMIENTO DEL SOFTWARE

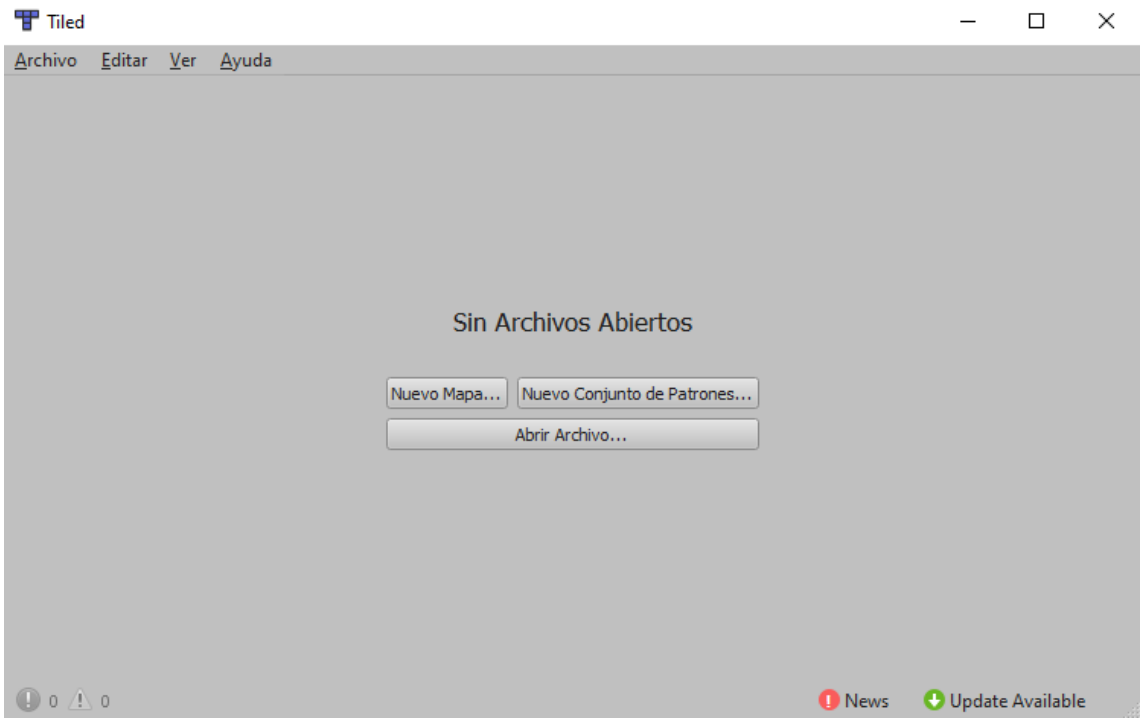
3.1. TILED

Para instalar tiled vamos a la página https://thorbjorn.itch.io/tiled y damos click en *"Download Now"*

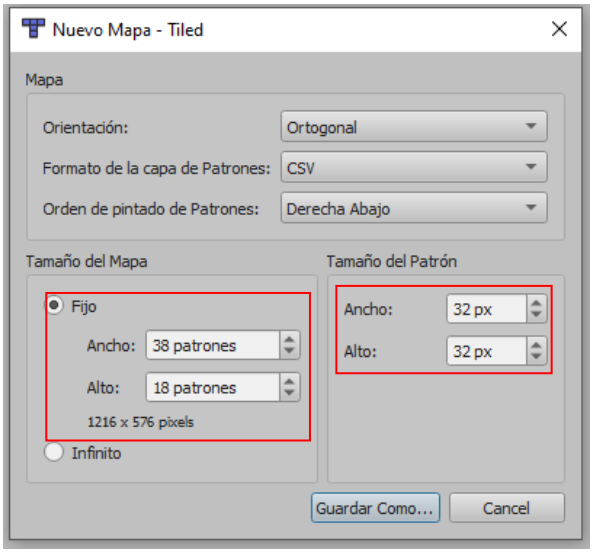


Nos dan la opción de donar al creador del software.

Una vez instalado el software lo abrimos

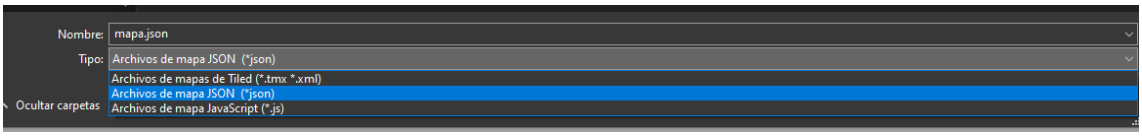


Damos click en nuevo mapa



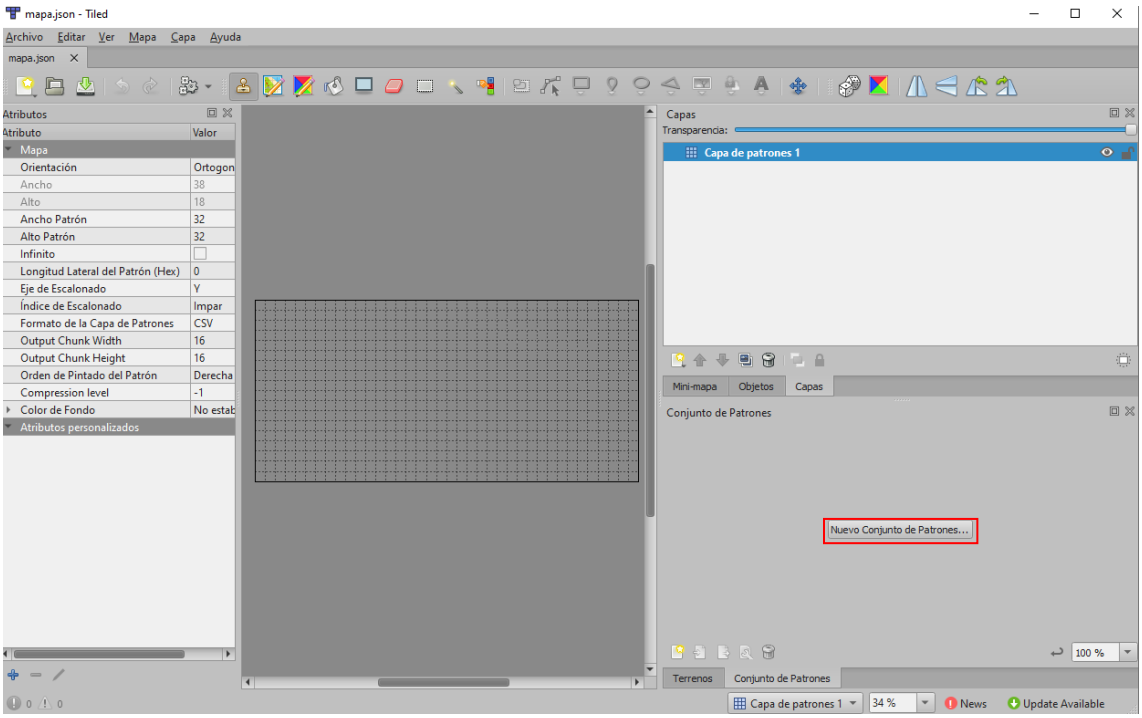
Indicamos el tamaño del mapa en patrones, (en mi caso 38x17) y el tamaño del patrón (tile) 32px por 32px.

Damos a *"Guardar como"*

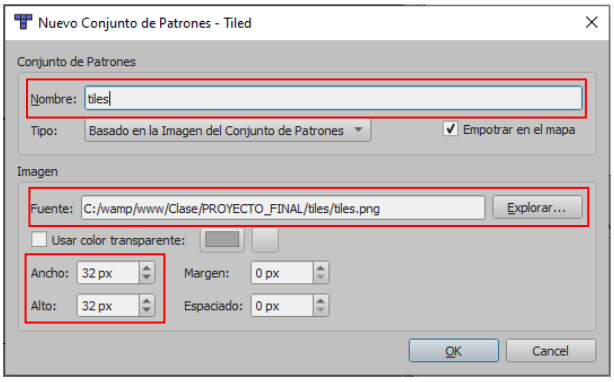


Damos un nombre y lo guardamos como fichero *".json"*

Una vez aquí damos click en *"Nuevo conjunto de patrones"*

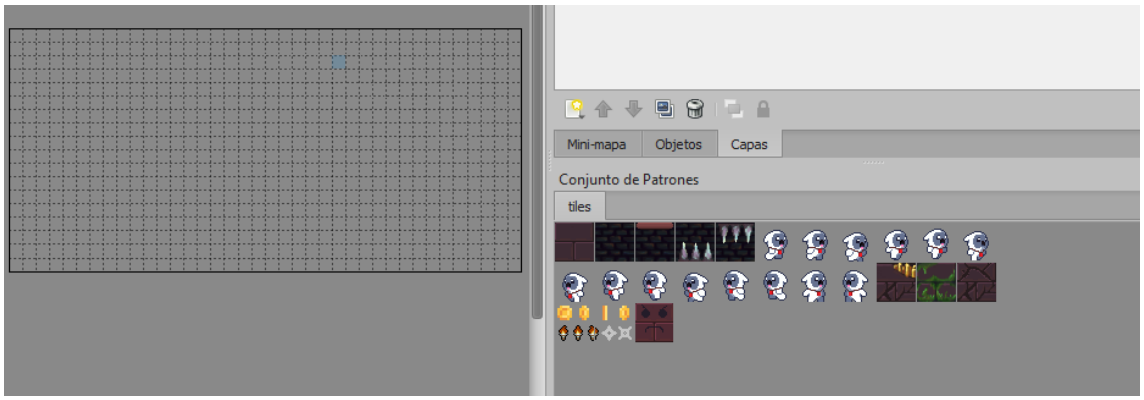


Damos click en “Nuevo conjunto de patrones”

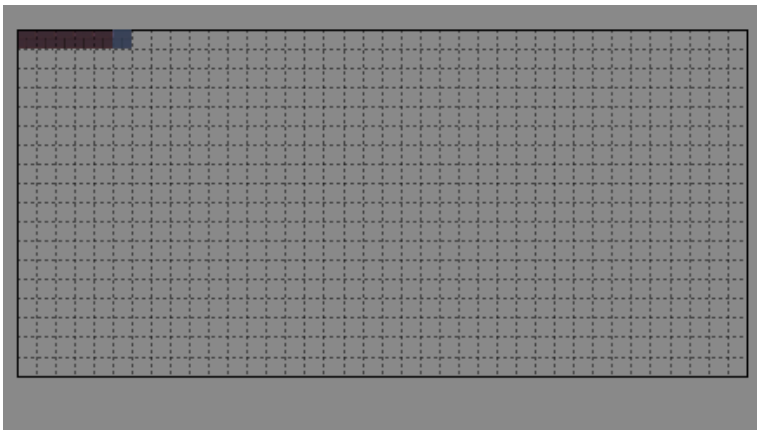


Seleccionamos nuestro tileset en el apartado “fuentes”, modificamos el ancho y el alto para que tengan el tamaño de nuestros tiles, (en mi caso 32x32), y le damos un nombre.

Una vez aquí seleccionamos una *tile* en la ventana de la derecha y hacemos click en la parte del mapa donde queramos que se dibuje.

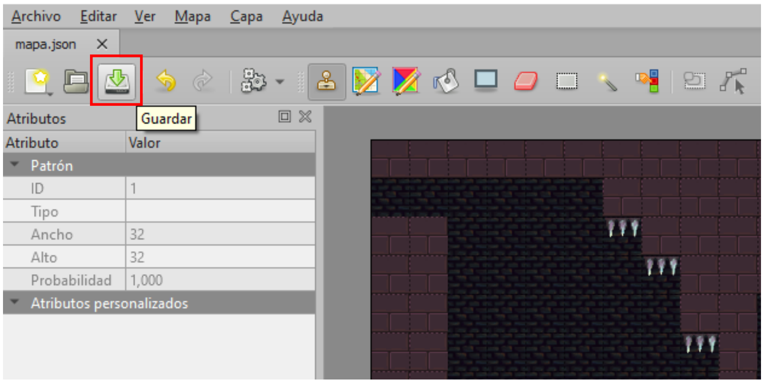


Podemos arrastrar el click para pintar sobre varias zonas del mapa.

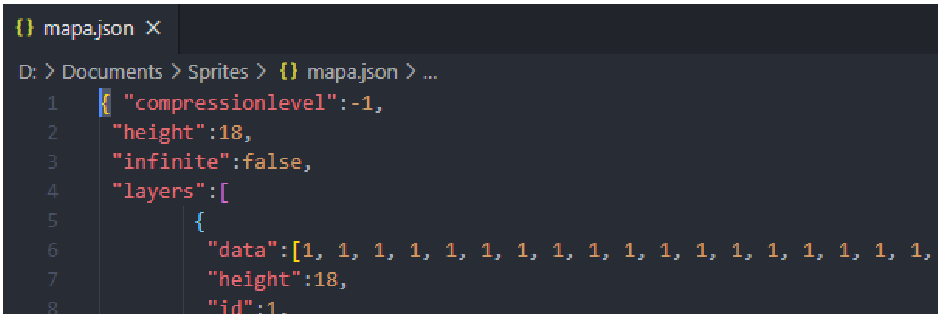


Tenemos también herramientas como en paint para rellenar una zona, borrar, mover, incluso randomizar cuando dibujamos.

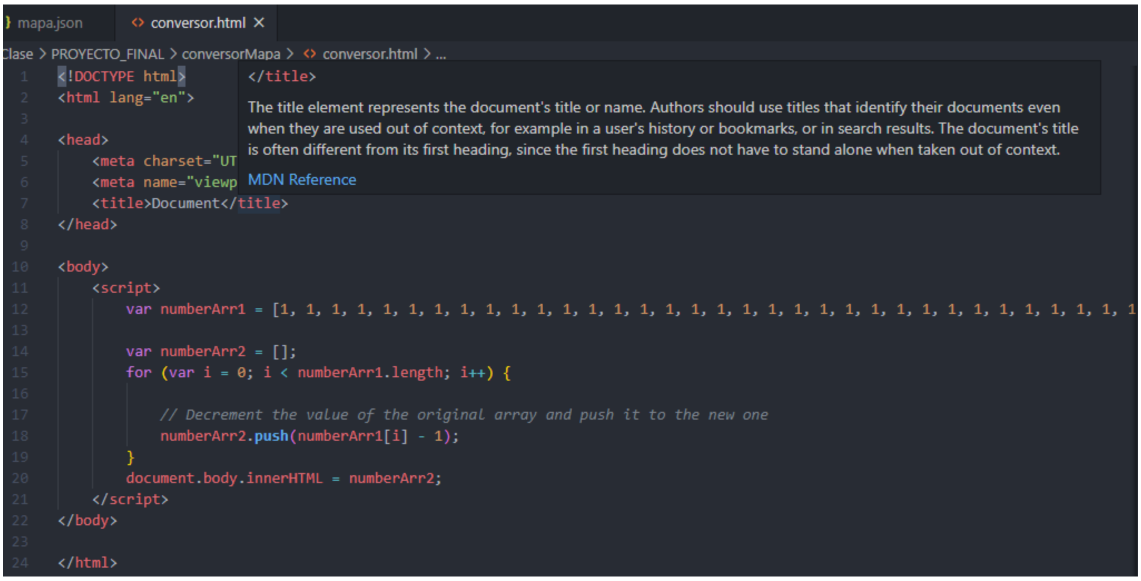
Una vez dibujado el mapa damos click en *"Guardar"*



Abrimos nuestro fichero *.json*



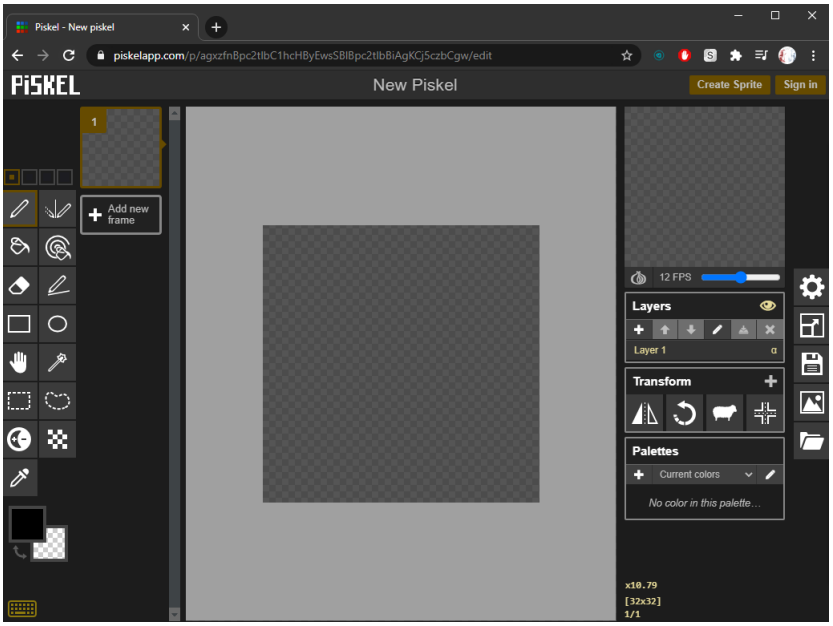
De esta información la que nos interesa es el array llamado "data", el cual contendrá un array con el mapa. El problema es que nos escribe el array con los valores +1, por lo que he creado un pequeño html que reduce en 1 todos los valores de un array:



Cambiamos la variable numberArr1 por nuestro array y cuando entremos en el html tendremos el array con los valores correctos para nuestro juego.

3.2. PISKEL

Piskel es un software online para dibujar pixelart por lo que no necesitaremos instalar nada, para ello vamos a https://www.piskelapp.com/ y damos click en *"Create Sprite"*

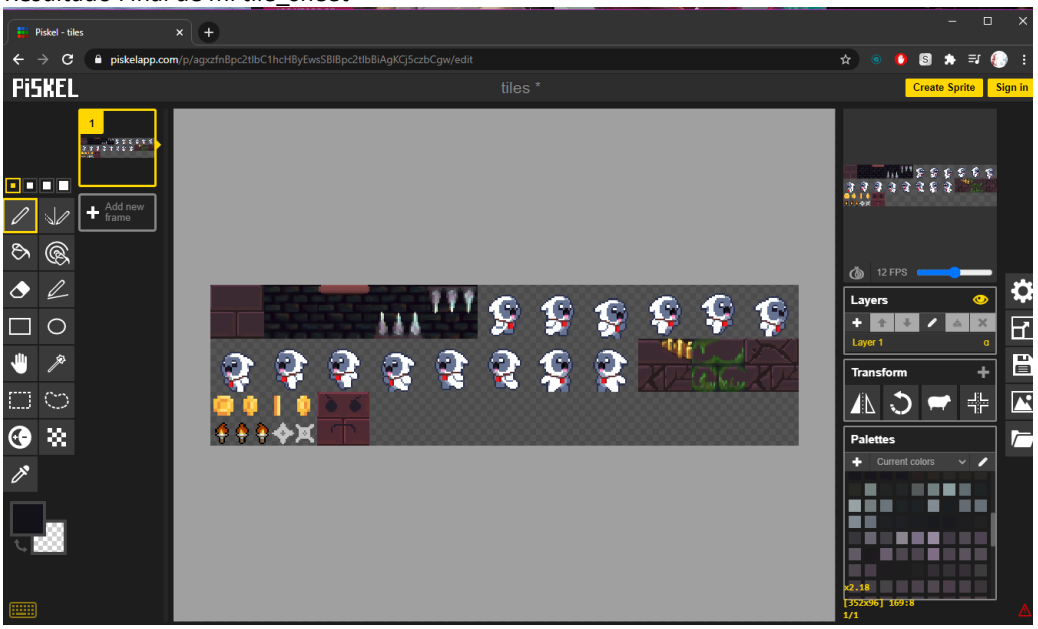


En la parte derecha podemos seleccionar capas para probar animaciones, transformar objetos y crear paletas de colores.

En el menú más pequeño de la derecha tenemos opciones para guardar, importar, exportar y reescalar.

En la parte izquierda tenemos herramientas para dibujar como el lápiz, la herramienta de relleno de selección etc.

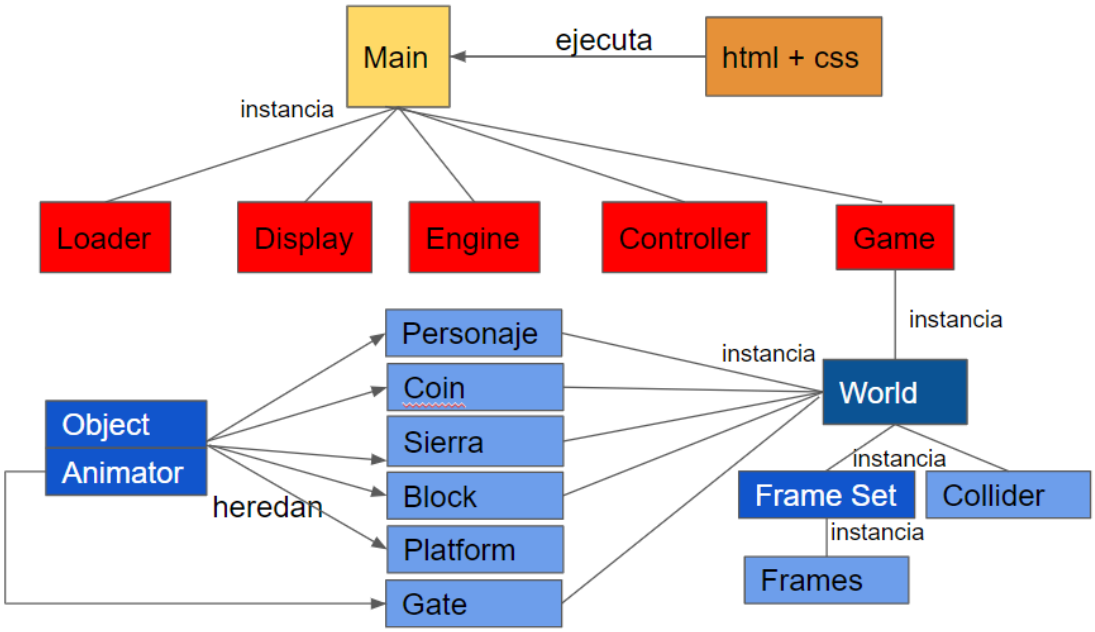
Resultado Final de mi tile_sheet



4. DIAGRAMA DE OBJETOS

En este apartado voy a explicar brevemente la estructura del código para tener una idea general sobre el funcionamiento del mismo.

Para ello he diseñado el siguiente esquema:



Veamos los objetos uno por uno:

- **Script Main:** Se ocupa de instanciar el resto de los objetos, así como la comunicación entre ellos.
- **Objeto *Loader*:** Se encarga de cargar tanto los niveles como las imágenes y los tiles (celdas gráficas).
- **Objeto *Engine*:** Se encarga del requestAnimationFrame de nuestro canvas junto con un método start para iniciar y un método stop para parar el juego. También hay código para mejorar la caída de frames para equipos menos potentes.
- **Objeto *Controller*:** Se encarga de recoger los inputs del usuario para trasladarlo al movimiento del personaje.
- **Objeto *Game*:** Se encarga de definir todo lo relacionado con el Juego en sí, es decir, el mundo, las animaciones, los objetos y las colisiones.

  El objeto *World* instancia al resto de objetos ya sea directa o indirectamente menos a los objetos *Object y Animator*.

  Los objetos *Personaje, Coin, Sierras, Block, Platform y Gate* se instancian haciendo una llamada a la clase object y animator (gates solo llama a object), y heredan sus métodos y propiedades.
- **Objeto *Display*:** Se encarga del reescalado y del dibujado del mapa y animaciones. Básicamente se encarga de toda la parte visual

5. DESARROLLO

Para el desarrollo del código voy a ir parte por parte explicando un poco por encima los distintos objetos junto con una captura del código. Vamos a ir en orden de carga de cada script.

5.1. CONTROLLER.JS

En este script como comenté anteriormente la utilizo para recoger la pulsación de las flechas izquierda, derecha y superior. Lo que hago es con un EventListener en main.js recojo las teclas que pulsa el jugador y en el método keyDownUp las parseo.

El objeto ButtonInput comprueba si esta pulsada o no la tecla

5.2. DISPLAY.JS

En este objeto tenemos un bloque de código para eliminar el contenido del body y crear la interfaz superior donde coloco las vidas, las monedas restantes y un botón para volver a la página index donde están las instrucciones del juego, por lo que voy a omitir la explicación de esta parte del código.

Tenemos el constructor del objeto display y sus métodos.

El método *dibujaMapa* que recoge la imagen con los tiles, las columnas de la imagen las columnas del mapa y los valores del mapa los cuales vamos a pasarles a partir de un archivo json.

Lo que hacemos es, recorrer el mapa completo y según el valor de cada elemento del mapa dibujamos un tile u otra.

Recogemos el punto exacto de recorte en valores x e y con *origen_x y origen_y* y recogemos donde vamos a dibujar este tile en valores x e y con *destino_x y destino_y*

Haciendo uso de la función *drawImage* posicionamos el tile en el canvas

```
this.dibujaMapa = function(img, img_col, mapa, m_col, tile_size) {
        for (let i = mapa.length - 1; i > -1; --i) {
            //recogemos el valor
            var tile = mapa[i];
            //recogemos el tile
            var origen_x = (tile % img_col) * tile_size;
            var origen_y = Math.floor(tile / img_col) * tile_size;
            //posicion donde vamos a dibujar el tile
            var destino_x = (i % m_col) * tile_size;
            var destino_y = Math.floor(i / m_col) * tile_size;
            //pintamos la tile en el buffer
            this.buffer.drawImage(img, origen_x, origen_y, tile_size, tile_size, destino_x, destino_y, tile_size, tile_size);
        }

    }
```

Es un poco complejo de explicar, pero creo que con un ejemplo básico podemos visualizarlo. Pongamos que tenemos un mapa [0,1,0,1] y estos dos tiles:

El mapa tiene 2 filas y 2 columnas y los tiles serán de 32px x 32px.

Para el primer valor del mapa (i=0) tendríamos origen_x = 0, origen_y = 0 (señalado con el cuadrado de color rojo)

Para el segundo valor (i=1) tendríamos origen_x=32, origen_y=0

Se repite para el tercer (i=2) y cuarto valor (i=3).

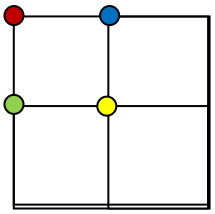
Ahora vamos con el dibujado en el canvas (el cavas está representado por el cuadrado blanco)

Para el primer valor del mapa (i=0) destino_x sería 0 y destino y sería 0 (rojo)

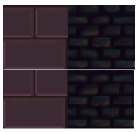
Para el segundo valor del mapa (i=1) destino_x sería 32 y destino y sería 0 (azul)

Para el primer valor del mapa (i=2) destino_x sería 0 y destino y sería 32 (verde)

Para el primer valor del mapa (i=0) destino_x sería 32 y destino y sería 32 (amarillo)



El resultado final sería el siguiente:



El método dibujaObjeto llama a drawImage al igual que dibujamapa pero pasándole los valores directamente, la utilizo para dibujar los distintos elementos del juego

```
    //funcion de dibujar el personaje
this.dibujaObjeto = function(img, origen_x, origen_y, destino_x, destino_y, width, height) {

    this.buffer.drawImage(img, origen_x, origen_y, width, height, Math.round(destino_x), Math.round(destino_y), width, height);
}
```

El método resize se encarga de el reescalado del canvas

```
this.resize = function(width, height, ratio) {
    //para que se reescale correctamente tenemos que comprobar que el ratio de altura : anchura sea el mismo
    if (height / width > ratio) {
        this.context.canvas.height = width * ratio;
        this.context.canvas.width = width;

    } else {
        this.context.canvas.height = height;
        this.context.canvas.width = height / ratio;
    }
    //para que se vean bien en pixel art y no aparezca borroso tras escalado los pixeles
    this.context.imageSmoothingEnabled = false;

}
```

El método gameOver pinta la imagen de cuando perdemos todas las vidas del tamaño del canvas

```
this.gameOver = function(img) {
    this.context.drawImage(img, 0, 0, this.buffer.canvas.width, this.buffer.canvas.height, 0, 0, this.context.canvas.width, this.context.canvas.height);
}
```

5.3. ENGINE.JS

El objeto Engine se encarga del request animation frame típico de los canvas pero con creado como un fixed timestep loop, esto se encarga de transformar el movimiento a frames reales de un videojuego. Esto es muy largo de explicar por lo que dejo un enlace en el que creo que lo explican mejor de lo que yo podría: https://www.gamedev.net/forums/topic/638521-fixed-time-step-game-loop/

```
const Engine = function(time_step, update, render) {

    this.accumulated_time = 0;
    this.animation_frame_request = undefined;
    this.time = undefined;
    this.time_step = time_step;
    this.updated = false;
    this.update = update;
    this.render = render;

    this.run = function(time_stamp) {
        this.animation_frame_request = window.requestAnimationFrame(this.handleRun);

        this.accumulated_time += time_stamp - this.time;
        this.time = time_stamp;

        if (this.accumulated_time >= this.time_step * 3) {
            this.accumulated_time = this.time_step;
        };

        while (this.accumulated_time >= this.time_step) {
            this.accumulated_time -= this.time_step;
            this.update(time_stamp);
            this.updated = true;
        };

        if (this.updated) {
            this.updated = false;
            this.render(time_stamp);
        };

    };
    this.handleRun = (time_step) => { this.run(time_step); };
}

Engine.prototype = {

    constructor: Engine,

    start: function() {
        this.accumulated_time = this.time_step;
        this.time = window.performance.now();
        this.animation_frame_request = window.requestAnimationFrame(this.handleRun);
    },

    stop: function() { window.cancelAnimationFrame(this.animation_frame_request); }
};
```

Las funciones que nos interesan sobre todo de aquí son las funciones de *start* la cual crea un requestAnimationFrame y *stop* la cual realiza un cancelAnimationFrame parando el juego.

5.4. LOADER.JS

El objeto Loader se encarga de recoger las imágenes y los ficheros json donde almacenamos información de los niveles

```
//clase que carga los archivos especificados

const Loader = function() {
    this.tile_set_image = undefined;
}

Loader.prototype = {
    constructor: Game.Loader,

    //rqJSON carga el fichero json de la url indicada, una vez cargado llama a la funcion callback especificada
    rqJSON: function(url, funcion) {
        var peticion = new XMLHttpRequest();
        peticion.addEventListener("load", function() {
            funcion(JSON.parse(this.responseText));
        }, {
            once: true,
        })
        peticion.open("GET", url);
        peticion.send();
    },

    //rqTileImage carga la image de los tiles según la url indicada, una vez cargada llama a la funcion callback especificada
    rqTileImage: function(url, funcion) {
        let imagen = new Image();
        imagen.addEventListener("load", function() {
            funcion(imagen);
        }, { once: true })
        imagen.src = url;
    }
}
```

Lo que hacemos es crear dos métodos con los parámetros de url y función y creamos un eventlistener para que cuando hayamos recogido el archivo (JSON o imagen) ejecute la función que le pasamos (normalmente a esto se le llama callback)

5.5. GAME.JS

El objeto Game se encarga de todo lo relacionado con el juego en sí, vamos a ir viendo los distintos objetos una por una

Constante Game la cual crea un objeto mundo y setea su metodo update como el metodo update del mundo

```
const Game = function() {
    this.world = new Game.World();
    this.update = function() {
        this.world.update();
    }
}

Game.prototype = {
    constructor: Game
}
```

Objeto TileSet

Define el tamaño y las columnas de nuestro tile set. También definimos un array con los distintos frames de las animaciones de los objetos creando objetos Frame que solo contendran los valores de x, y, width y height para recortar las imagenes

```
////////////////////////////////////////////////////////////OBJETO TILE SET//////////////////////////////////////////////////////////////////////
Game.TileSet = function(tile_size, columnas) {

    this.tile_size = tile_size;
    this.columnas = columnas;

    var frame = Game.Frame;
    this.array_frames = [
        //mirando derecha [0]
        new frame(196, 38, 21, 26),
        //moviendose dcha [1-6] el frame [4] equivale al salto derecha.
        new frame(166, 6, 21, 26), new frame(197, 6, 21, 26), new frame(229, 6, 21, 26), new frame(262, 6, 21, 26), new frame(294, 6, 21, 26), new frame(325, 6, 21, 26),
        //mirando izq [7]
        new frame(229, 38, 21, 26),
        //moviendose izq [8-14], frame [10] equivale al salto izquierda
        new frame(166, 38, 21, 26), new frame(134, 38, 21, 26), new frame(102, 38, 21, 26), new frame(70, 38, 21, 26), new frame(38, 38, 21, 26), new frame(6, 38, 21, 26),
        //monedas
        new frame(0, 64, 16, 16), new frame(16, 64, 16, 16), new frame(32, 64, 16, 16), new frame(48, 64, 16, 16),
        //sierras
        new frame(37, 82, 13, 13), new frame(50, 82, 13, 13),
        //blocks
        new frame(64, 64, 32, 32),
        //antorchas
        new frame(3, 82, 8, 13), new frame(15, 82, 8, 13), new frame(17, 82, 8, 13)
    ];
}
Game.TileSet.prototype = { constructor: Game.TileSet };

//para definir los frames dentro de los arrays de tiles para cada animacion
Game.Frame = function(x, y, width, height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
}
Game.Frame.prototype = { constructor: Game.Frame };
```

Objeto World

Aquí instanciamos las distintas propiedades del mundo como son la fricción o la gravedad e instanciamos algunos objetos como el colisionador, el personaje y el objeto TileSet

```
Game.World = function(friccion = 0.15, gravedad = 2) {

    //variables de nivel
    this.id_nivel = "1";
    this.score = 0;

    //variables del mapa
    this.friccion = friccion;
    this.gravedad = gravedad;
    this.columnas = 36;
    this.filas = 17;

    //instanciacion de objetos
    this.tile_set = new Game.TileSet(32, 11);
    this.collider = new Game.Collider();
    this.personaje = new Game.Personaje(35, 35);

    //""reescalado"""
    this.height = this.tile_set.tile_size * this.filas;
    this.width = this.tile_set.tile_size * this.columnas;
}
```

El método cargaNivel inicializa el resto de los objetos que tenemos en el nivel y que pasamos por JSON como son el mapa, las monedas, las puertas de cambio de nivel y los distintos enemigos.

```
Game.World.prototype = {

    constructor: Game.World,
    //metodo para cargar niveles a partir de jsons
    cargarNivel: function(nivel) {
        this.mapa = nivel.mapa;
        this.columnas = nivel.columnas;
        this.filas = nivel.filas;
        this.gates = new Array();
        this.coins = new Array();
        this.sierras = new Array();
        this.blocks = new Array();
        this.score = nivel.coins.length + 1;
        this.id_nivel = nivel.id_nivel;
        this.personaje.setReaparicionX(nivel.reaparicion_x * this.tile_set.tile_size);
        this.personaje.setReaparicionY(nivel.reaparicion_y * this.tile_set.tile_size);
        //padding para centrar los objetos
        var padding = this.tile_set.tile_size / 4;
```

Algunos objetos se instancian de forma dinámica con arrays (monedas, sierras, puertas y bloques)

```
//seteamos las monedas
for (var j = nivel.coins.length - 1; j > -1; j--) {
    var moneda = nivel.coins[j];
    this.coins[j] = new Game.Coin(moneda[0] * this.tile_set.tile_size + padding, moneda[1] * this.tile_set.tile_size + padding + 4);
}

//seteamos las sierras
for (var k = nivel.sierras.length - 1; k > -1; k--) {
    var sierra = nivel.sierras[k];
    this.sierras[k] = new Game.Sierra(sierra[0] * this.tile_set.tile_size - 7, sierra[1] * this.tile_set.tile_size + padding + 1, sierra[2], sierra[3], sierra[4], sierra[5]);
}

//seteamos las plataformas
for (var m = nivel.platforms.length - 1; m > -1; m--) {
    var plataforma = nivel.platforms[m];
    this.platforms[m] = new Game.Platform(plataforma[0] * this.tile_set.tile_size - 7, plataforma[1] * this.tile_set.tile_size + padding + 1, plataforma[2], plataforma[3], plataforma[4], plataforma[5]);
}

//seteamos las puertas
for (var i = 0; i < nivel.gates.length; i++) {
    var gate = nivel.gates[i];
    this.gates[i] = new Game.Gate(gate);
}

//seteamos los bloques
for (var l = 0; l < nivel.blocks.length; l++) {
    var block = nivel.blocks[l];
    this.blocks[l] = new Game.Block(block[0] * this.tile_set.tile_size, block[1] * this.tile_set.tile_size, block[2] * this.tile_set.tile_size, block[1] * this.tile_set.tile_size + block[3] * this.tile_set.tile_size);
}
```

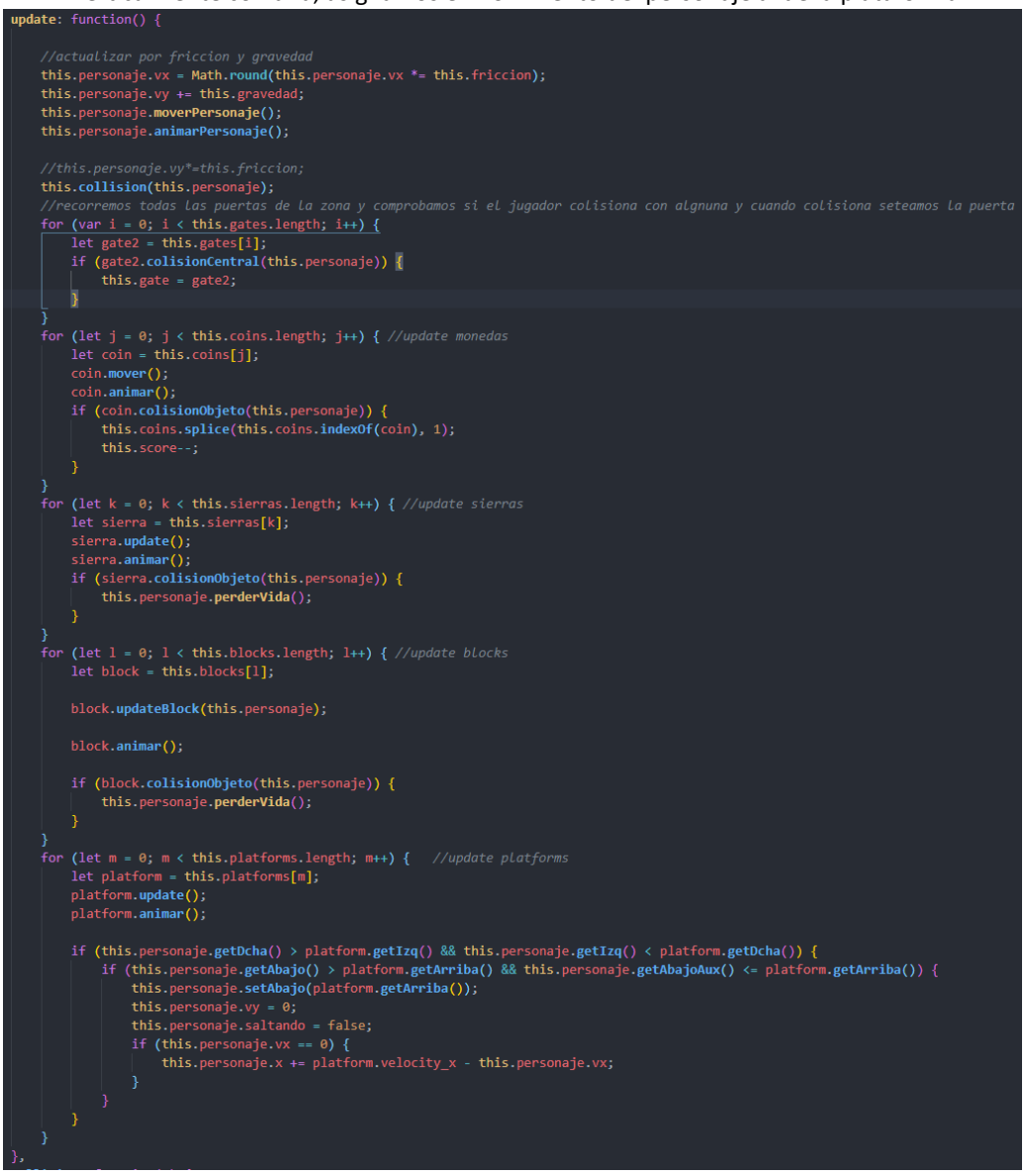
En este método también compruebo si el jugador ha entrado en alguna puerta le reposiciono en el mapa y reseteo el atributo de puerta

```
//si el jugador ha entrado en la puerta
if (this.gate) {
    //seteamos la posicion del personaje (tanto x como y) en la posicion de destino de la puerta
    if (this.gate.destino_x != -1) {
        this.personaje.setCentroX(this.gate.destino_x);
        this.personaje.setCentroXAux(this.gate.destino_x);
    }
    if (this.gate.destino_y != -1) {
        this.personaje.setCentroY(this.gate.destino_x);
        this.personaje.setCentroYAux(this.gate.destino_y);
    }
    //reseteamos la puerta para no entrar en bucle infinito
    this.gate = undefined;
}
```

any

El método update se ejecuta en cada frame del juego y se encarga de:

- Modificar la velocidad del personaje con la gravedad y la fricción para que el personaje tenga gravedad y una fricción (la fricción se encarga de "deslizar" el personaje cuando deje de moverse)
- Mover y animar los distintos objetos del juego
- Comprobar las colisiones del personaje con el mundo
- Comprobar si el jugador ha colisionado con una de las puertas y, en caso de colisionar centralmente, instancia la puerta con la que ha colisionado (se utiliza después para que cuando se haya instanciado una puerta se cambie al nivel que indique la puerta).
- El movimiento, las animaciones y las colisiones para cada moneda con el jugador que en caso de colisionar resta 1 a la variable score la cual se encarga de contar las monedas restantes en el juego
- El movimiento, las animaciones y las colisiones para cada obstáculo (sierra y bloques) y, en caso de que el personaje colisione con uno de ellos, ejecutamos el método perderVida
- El movimiento y las animaciones para cada plataforma, que, en caso de colisionar verticalmente con una, asignamos el movimiento del personaje al de la plataforma



```
update: function() {

    //actualizar por friccion y gravedad
    this.personaje.vx = Math.round(this.personaje.vx *= this.friccion);
    this.personaje.vy += this.gravedad;
    this.personaje.moverPersonaje();
    this.personaje.animarPersonaje();

    //this.personaje.vy*=this.friccion;
    this.collision(this.personaje);
    //recorremos todas las puertas de la zona y comprobamos si el jugador colisiona con algnuna y cuando colisiona seteamos la puerta
    for (var i = 0; i < this.gates.length; i++) {
        let gate2 = this.gates[i];
        if (gate2.colisionCentral(this.personaje)) {
            this.gate = gate2;
        }
    }
    for (let j = 0; j < this.coins.length; j++) { //update monedas
        let coin = this.coins[j];
        coin.mover();
        coin.animar();
        if (coin.colisionObjeto(this.personaje)) {
            this.coins.splice(this.coins.indexOf(coin), 1);
            this.score--;
        }
    }
    for (let k = 0; k < this.sierras.length; k++) { //update sierras
        let sierra = this.sierras[k];
        sierra.update();
        sierra.animar();
        if (sierra.colisionObjeto(this.personaje)) {
            this.personaje.perderVida();
        }
    }
    for (let l = 0; l < this.blocks.length; l++) { //update blocks
        let block = this.blocks[l];

        block.updateBlock(this.personaje);

        block.animar();

        if (block.colisionObjeto(this.personaje)) {
            this.personaje.perderVida();
        }
    }
    for (let m = 0; m < this.platforms.length; m++) {   //update platforms
        let platform = this.platforms[m];
        platform.update();
        platform.animar();

        if (this.personaje.getDcha() > platform.getIzq() && this.personaje.getIzq() < platform.getDcha()) {
            if (this.personaje.getAbajo() > platform.getArriba() && this.personaje.getAbajoAux() <= platform.getArriba()) {
                this.personaje.setAbajo(platform.getArriba());
                this.personaje.vy = 0;
                this.personaje.saltando = false;
                if (this.personaje.vx == 0) {
                    this.personaje.x += platform.velocity_x - this.personaje.vx;
                }
            }
        }
    }
},
```

Objeto Collider

Este objeto se encarga de asignar colisiones según el tile del mapa que sea.

```
Game.Collider = function() {
        //colisiones dependiendo del bloque que sea
        this.collide = function(value, obj, tile_x, tile_y, tile_size) {
            switch (value) {
                case 0:
                    if (this.colisionSuperior(obj, tile_y)) return;
                    if (this.colisionIzq(obj, tile_x)) return;
                    if (this.colisionDcha(obj, tile_x + tile_size)) return;
                    this.colisionInferior(obj, tile_y + tile_size);
                    break;
                case 2:
                    this.colisionSuperior(obj, tile_y);
                    break;
                case 3:
                    this.colisionPinchosSup(obj, tile_y + tile_size / 2.5)
                    break;
                case 4:
                    this.colisionPinchosInf(obj, tile_y + (tile_size / 2.5))
                    break;
                case 20:
                    this.colisionDcha(obj, tile_x + tile_size);
                    break;
                case 21:
                    this.colisionIzq(obj, tile_x);
                    break;

            }
        }
    }
```

Voy a omitir la explicación de los métodos colisionSuperior, Inferior, Izquierda, Derecha, Pinchos superior y pinchos inferior puesto que es bastante repetitivo, simplemente comprueba que si el objeto que le pasamos (jugador) colisiona con la izquierda (x), derecha (x+width), arriba (y) o debajo (y+height) del tile dependiendo del tile que sea y si colisiona le seteo su velocidad a 0 y su posicionamiento pegado a la tile que haya colisionado.

En el caso de la colisión con los pinchos simplemente cuando colisiona superior o inferiormente (asignado a los pinchos del techo y del suelo respectivamente) llamamos al método perderVida el cual explicaré más adelante

Objeto Animator

Este objeto es el encargado de dibujar y animar los distintos objetos que definamos.

El método setFrame asigna un array de tiles (lo utilizaremos para cambiar las animaciones del jugador dependiendo del movimiento)

El método animar es una función que comprueba el valor booleano *loop* para cuando queramos recorrer el array de animaciones en bucle.

El método animación recorre el array de animaciones en bucle y con la variable delay podemos indicar la velocidad a la que queramos que se “anime” el objeto (velocidad en la que recorre el array).

```
Game.Animator = function(frame_set, delay) {
    this.count = 0;
    if (delay >= 1) {
        this.delay = delay;
    } else { delay = 1; }
    this.frame_set = frame_set;
    this.f_index = 0;
    this.f_value = frame_set[0];
    this.loop = true;
}

Game.Animator.prototype = {
    constructor: Game.Animator,

    //
    animar: function() {
        if (this.loop) {
            this.animacion();
        }
    },

    //setear el nuevo array de frames
    setFrame(frame_set, loop, delay = 10, f_index = 0) {
        this.count = 0;
        this.delay = delay;
        this.frame_set = frame_set;
        this.f_index = f_index;
        this.f_value = frame_set[f_index];
        this.loop = loop;
    },

    //funcion para recorrer el array de frames en bucle para los movimientos horizontales
    animacion: function() {
        this.count++;
        while (this.count > this.delay) {
            this.count -= this.delay;
            if (this.f_index < this.frame_set.length - 1) {
                this.f_index++;
            } else {
                this.f_index = 0;
            }
            this.f_value = this.frame_set[this.f_index];
        }
    }

}
```

Objeto Personaje:

La propiedad frames define los distintos frames para cada “estado”del personaje (moviéndose izquierda y derecha, saltando izquierda y derecha y estando quieto izquierda y derecha)

Los métodos setReaparicionX y setReaparicióny sirven para definir la posición del personaje cuando pierda una vida

El método saltar comprueba si no está saltando (para no saltar infinitamente) y le asigna una velocidad negativa para el salto del personaje

```
Game.Personaje = function(x, y) {
    this.reaparicion_x = 0;
    this.reaparicion_y = 0;
    Game.Object.call(this, x, y, 21, 25);
    Game.Animator.call(this, Game.Personaje.prototype.frames["dcha"]);
    this.vx = 0;
    this.vy = 0;
    this.saltando = true;
    this.ladomira = 1;
    this.vidas = 2;
}

//funciones personaje
Game.Personaje.prototype = {
    constructor: Game.Personaje,

    //arrays para las animaciones
    frames: {
        //arrays animaciones hacia la derecha
        "dcha": [0],
        "m_dcha": [1, 2, 3, 4, 5, 6],
        "s_dcha": [4],
        //arrays animaciones hacia la izquierda
        "izq": [7],
        "m_izq": [8, 9, 10, 11, 12, 13],
        "s_izq": [11],
    },
    setReaparicionX(i) {
        this.reaparicion_x = i;
    },
    setReaparicionY(i) {
        this.reaparicion_y = i;
    },
    //funcion saltar
    saltar: function() {
        if (!this.saltando) {
            this.saltando = true;
            this.vy -= 18;
        }
    },
```

El método mverIzq y moverDcha setea la dirección del personaje asignándole una velocidad positiva o negativa y asigna el lado en el que está mirando el personaje para la animación de este.

El método perder vida nos quita una vida y setea la posición del personaje en su posición de reaparición.

El método moverpersonaje simplemente le suma a las x e y sus velocidades correspondientes para el movimiento del personaje

El método animarPersonaje setea los frames de animación del personaje dependiendo de si está saltando o no, del lado en el que está orientando y de si están en movimiento o no.

```
    //movimiento izq
    moverIzq: function() {
        this.vx -= 50;
        this.ladomira = -1;
    },

    //movimiento dcha
    moverDcha: function() {
        this.vx += 50;
        this.ladomira = 1;
    },
    //funcion de perder vida
    perderVida: function() {
        this.x = this.reaparicion_x;
        this.y = this.reaparicion_y;
        this.vidas--;
        this.saltando = false;
        this.vx = 0;
        this.vy = 0;
    },

    moverPersonaje: function() {
        this.aux_x = this.x;
        this.aux_y = this.y;
        this.x += this.vx;
        this.y += this.vy;
    },

    //funcion de animar al personaje
    animarPersonaje: function() {
        //dependiendo del lado que mire se le asignan una animacion u otra así como dependiendo de si está saltando o no
        if (this.vy < 0) {
            if (this.ladomira < 0) {
                this.setFrame(this.frames["s_izq"], false);
            } else {
                this.setFrame(this.frames["s_dcha"], false);
            }
        } else if (this.ladomira < 0) {
            if (this.vx < -0.1) {
                this.setFrame(this.frames["m_izq"], true, 3);
            } else {
                this.setFrame(this.frames["izq"], false);
            }
        } else if (this.ladomira > 0) {
            if (this.vx > 0.1) {
                this.setFrame(this.frames["m_dcha"], true, 3);
            } else {
                this.setFrame(this.frames["dcha"], false);
            }
        }
        this.animar();
    }
}
```

Objeto Gate

Instancia las puertas en la posición x e y pasando un objeto puerta.

```
Game.Gate = function(gate) {
    Game.Object.call(this, gate.x, gate.y, gate.width, gate.height);
    this.destino_x = gate.destino_x;
    this.destino_y = gate.destino_y;
    this.nivel_destino = gate.nivel_destino;
}
Game.Gate.prototype = {}
Object.assign(Game.Gate.prototype, Game.Object.prototype);
Game.Gate.prototype.constructor = Game.Gate;
```

Objeto Coin

para instanciar las monedas en la posición x e y.

```
Game.Coin = function(x, y) {
    this.x = x;
    this.y = y;
    this.width = 16;
    this.height = 16;
    Game.Object.call(this.x, this.y, this.width, this.height);
    Game.Animator.call(this, Game.Coin.prototype.frame_sets["monedas"], 10);
    this.valor_movimiento = 0;
    this.f_index = 0;
    this.vectory = 0;
}

Game.Coin.prototype = {
    frame_sets: {
        "monedas": [14, 15, 16, 17]
    },
    mover: function() {
        this.vectory -= 0.1;
        this.y += Math.sin(this.vectory) * 0.4;
    }
}
Object.assign(Game.Coin.prototype, Game.Object.prototype);
Object.assign(Game.Coin.prototype, Game.Animator.prototype);

Game.Coin.prototype.constructor = Game.Coin;
```

El método mover hace que las monedas suban y bajen en el eje Y y frame_sets define el array de animación de las monedas.

Objeto Sierra

Instancia las distintas sierras del juego en la posición x e y

```
Game.Sierra = function(x, y, orientacion, radio, p, v) {
    this.x = x;
    this.aux_x = x;
    this.aux_y = y;
    this.y = y;
    this.v = v;
    this.width = 13;
    this.height = 13;
    this.radio = radio;
    this.orientacion = orientacion;
    Game.Object.call(this.x, this.y, this.width, this.height);
    Game.Animator.call(this, Game.Sierra.prototype.frame_sets["sierras"], 2);
    this.vx = 0;
    this.vy = 0;
    this.p = p;
}
Game.Sierra.prototype = {
    frame_sets: {
        "sierras": [18, 19]
    },
    update: function() {
        //1 es horizontal 0 vertical
        if (this.orientacion == 1) {
            if (this.p == 0) {
                this.vx += this.v
            } else if (this.p == 1) {
                this.vx -= this.v
            }
            this.x = this.aux_x + Math.sin(this.vx) * this.radio;
        } else if (this.orientacion == 0) {
            if (this.p == 0) {
                this.vy += this.v
            } else if (this.p == 1) {
                this.vy -= this.v
            }
            this.y = this.aux_y + Math.sin(this.vy) * this.radio;
        }
    }
}

Object.assign(Game.Sierra.prototype, Game.Object.prototype);
Object.assign(Game.Sierra.prototype, Game.Animator.prototype);

Game.Sierra.prototype.constructor = Game.Sierra;
```

Frame_sets define el array de animación de las sierras

El método update se encarga del movimiento de las sierras, que dependiendo de las variables orientación, radio, p y v dibuja en el mapa una sierra con orientación vertical u horizontal (variable orientación) que empieza moviéndose hacia arriba o abajo (variable p) en un radio definido (variable radio) y con una velocidad definida (variable v)

Objeto Block

Instancia los bloques en la posición x e y

```
Game.Block = function(x, y, y_inicial, y_final) {
    this.x = x;
    this.y = y;
    this.y_inicial = y_inicial;
    this.y_final = y_final;
    this.state = "quieto";
    this.vy = 0;
    this.width = 31.5;
    this.height = 31.5;
    Game.Object.call(this.x, this.y, this.width, this.height);
    Game.Animator.call(this, Game.Block.prototype.frame_sets["bloque"]);

}
Game.Block.prototype = {
    frame_sets: { "bloque": [20] },
    updateBlock: function(jugador) {
        switch (this.state) {
            case "quieto":
                if (jugador.getCentroX() < this.getDcha() && jugador.getCentroX() > this.getIzq() && this.y_final + 32 > jugador.getCentroY() && this.y_inicial + 32 < jugador.getCentroY()) {
                    this.state = "cae";
                }
                break;

            case "cae":
                this.vy = 6;
                this.y += this.vy;
                if (this.y > this.y_final) {
                    this.y = this.y_final;
                    this.vy = 0;
                    this.state = "elevarse";
                }
                break;
            case "elevarse":
                this.vy = 0.9;
                this.y -= this.vy;
                if (this.getArriba() < this.y_inicial) {
                    this.y = this.y_inicial;
                    this.vy = 0;
                    this.state = "quieto";
                }

        }
    }
}
```

La propiedad Frame_sets define el array de animación de las sierras

El método updateblock cambia el movimiento del bloque según el estado de este (por defecto se instancia en el estado quieto)

Según el estado del bloque tenemos 3 movimientos

- El estado quieto donde no se mueve, pero espera a que el personaje esté dentro de la zona donde cae el bloque en el eje y si el personaje entra dentro de esta zona cambiamos el estado del bloque a “cae”
- El estado cae hace que se mueva verticalmente hacia abajo y si llega a la variable y_final que define el tile hasta la cual el bloque cae, cambia su estado a “elevarse”
- El estado elevarse hace que el bloque se eleve hasta su posición inicial, una vez llega a la posición inicial cambia su estado a “quieto”

Objeto Platform

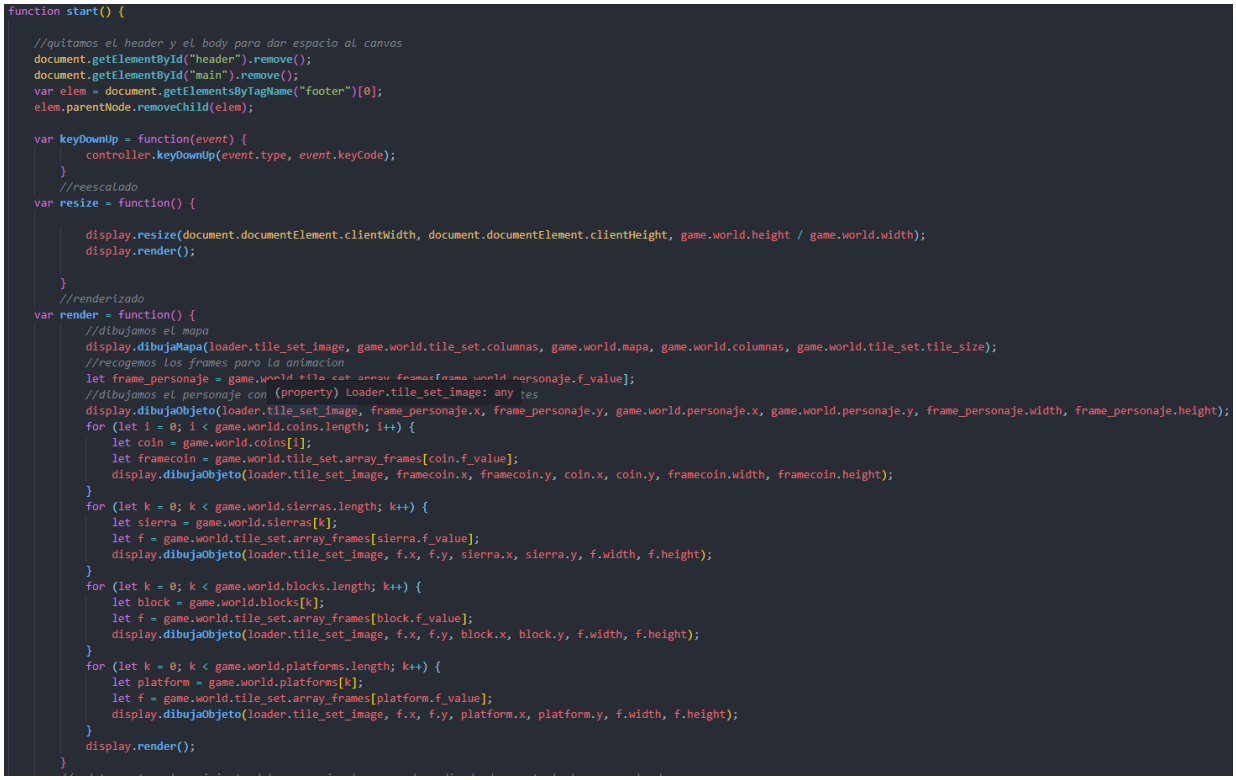
Instancia las plataformas que se mueven lateralmente

```
Game.Platform = function(x, y, radio, p, v) {
    this.x = x;
    this.aux_x = x;
    this.aux_y = y;
    this.y = y;
    this.v = v;
    this.width = 32;
    this.height = 32;
    this.velocity_x = 0;
    this_velocity_y = 0;
    this.radio = radio;
    Game.Object.call(this.x, this.y, this.width, this.height);
    Game.Animator.call(this, Game.Platform.prototype.frame_sets["Platforms"], 2);
    this.vx = 0;
    this.vy = 0;
    this.p = p;
    console.log(this.p);
}

Game.Platform.prototype = {
    frame_sets: {
        "Platforms": [22]
    },
    update: function() {
        //1 es horizontal 0 vertical
        if (this.p == 0) {
            this.vx += this.v;
        } else if (this.p == 1) {
            this.vx -= this.v;
        }
        this.velocity_x = this.aux_x + Math.sin(this.vx) * this.radio - this.x;
        this.x += this.velocity_x;
    }
}


Object.assign(Game.Platform.prototype, Game.Object.prototype);
Object.assign(Game.Platform.prototype, Game.Animator.prototype);
```

La función update se encarga del movimiento de las plataformas que dependiendo de la variable p que le pasemos con el fichero json, tendrá valor 1 o 0 y dependiendo del valor del mismo, se moverá primero hacia la izquierda o hacia la derecha.

5.6. MAIN.JS

El script main.js es el encargado de la instanciación de los distintos objetos y de la comunicación entre ellos



```
function start() {

    //quitamos el header y el body para dar espacio al canvas
    document.getElementById("header").remove();
    document.getElementById("main").remove();
    var elem = document.getElementsByTagName("footer")[0];
    elem.parentNode.removeChild(elem);

    var keyDownUp = function(event) {
            controller.keyDownUp(event.type, event.keyCode);
        }
        //reescalado
    var resize = function() {

            display.resize(document.documentElement.clientWidth, document.documentElement.clientHeight, game.world.height / game.world.width);
            display.render();

        }
        //renderizado
    var render = function() {
            //dibujamos el mapa
            display.dibujaMapa(loader.tile_set_image, game.world.tile_set.columnas, game.world.mapa, game.world.columnas, game.world.tile_set.tile_size);
            //recogemos los frames para la animacion
            let frame_personaje = game.world.tile_set.array_frames[game.world.personaje.f_value];
            //dibujamos el personaje con (property) Loader.tile_set_image: any tes
            display.dibujaObjeto(loader.tile_set_image, frame_personaje.x, frame_personaje.y, game.world.personaje.x, game.world.personaje.y, frame_personaje.width, frame_personaje.height);
            for (let i = 0; i < game.world.coins.length; i++) {
                let coin = game.world.coins[i];
                let framecoin = game.world.tile_set.array_frames[coin.f_value];
                display.dibujaObjeto(loader.tile_set_image, framecoin.x, framecoin.y, coin.x, coin.y, framecoin.width, framecoin.height);
            }
            for (let k = 0; k < game.world.sierras.length; k++) {
                let sierra = game.world.sierras[k];
                let f = game.world.tile_set.array_frames[sierra.f_value];
                display.dibujaObjeto(loader.tile_set_image, f.x, f.y, sierra.x, sierra.y, f.width, f.height);
            }
            for (let k = 0; k < game.world.blocks.length; k++) {
                let block = game.world.blocks[k];
                let f = game.world.tile_set.array_frames[block.f_value];
                display.dibujaObjeto(loader.tile_set_image, f.x, f.y, block.x, block.y, f.width, f.height);
            }
            for (let k = 0; k < game.world.platforms.length; k++) {
                let platform = game.world.platforms[k];
                let f = game.world.tile_set.array_frames[platform.f_value];
                display.dibujaObjeto(loader.tile_set_image, f.x, f.y, platform.x, platform.y, f.width, f.height);
            }
            display.render();
        }
```

Para empezar, tenemos una función start la cual hace que se ejecute el código cuando presionemos el botón de jugar en nuestra página, elimina el header y el contenido del body de nuestra página y carga el juego.

La función keyDownUp realiza una llamada al método controller para el seteo del movimiento de nuestro personaje

La función resize reescala el canvas llamando al método resize del objeto display para que tenga el tamaño del cliente y le pasamos como ratio el heigth/width del mundo y despues ejecuta el método render del objeto display

La función render se encarga llamar a los distintos métodos de dibujado como son el dibujado del mapa y el dibujado de los objetos (los cuales cambian con la animación que es lo que realizamos aquí)

La función update se manejar todos los métodos “update” de los distintos objetos de Game

Aquí se asigna el movimiento del jugador dependiendo de las teclas que hayamos pulsado las cuales están controlada por el método controller

Se dibujan en el cartel y se comprueba las vidas para que si nos quedamos sin vidas llame al método gameOver pasándole la imagen que recogemos del objeto loader.

Se dibujan en el cartel de monedas restantes las monedas que quedan por recoger

Se ejecuta el método update del objeto game, el cual ejecuta el método update del objeto world, el cual ejecuta el método “update” del resto de objetos en game.js (movimiento personaje, monedas etc.)

Se comprueba si se ha instanciado una puerta (se instancian solo cuando el personaje colisiona centralmente con una de ellas) y si está instanciada paramos el juego, se carga el siguiente nivel recogiendo el archivo json con el objeto loader y se inicia de nuevo el juego

```
var update = function() {
    if (controller.izq.active) {
        game.world.personaje.moverIzq();
    }
    if (controller.dcha.active) {
        game.world.personaje.moverDcha();
    }
    if (controller.saltar.active) {
        game.world.personaje.saltar();
        controller.saltar.active = false;
    }
    if (game.world.personaje.vidas < 0) {
        loader.rqTileImage("tiles/prueba.png", (imagen) => {
            display.gameOver(imagen);
            engine.stop();
        })
    }

    //añadimos el cartel de  vidas restantes
    if (game.world.personaje.vidas >= -1) {
        display.cartelvidas.innerHTML = "Vidas restantes: " + "&#10084;".repeat(game.world.personaje.vidas + 1);
    }
    if (game.world.score >= -1) {
        display.monedasrestantes.innerHTML = "Monedas restantes: " + "<img src='tiles/coin.png' width='18px' height='18px'> ".repeat(game.world.score);
    }
    game.update();
    //si el personaje colisiona con una puerta inizializa la variable gate la cual indica que nivel cargar
    if (game.world.gate) {
        //paramos el juego
        engine.stop();
        //recogemos el nuevo nivel y lo cargamos
        loader.rqJSON("json/nivel" + game.world.gate.nivel_destino + ".json", (nivel) => {
            //cargamos el nivel
            game.world.cargarNivel(nivel);
            //iniciamos el juego de nuevo
            engine.start();

        });
        return;
    }

}
```

En esta parte se instancian los objetos que hemos utilizado en las funciones anteriores, se recogen el nivel y el tile_set y se inicia el juego

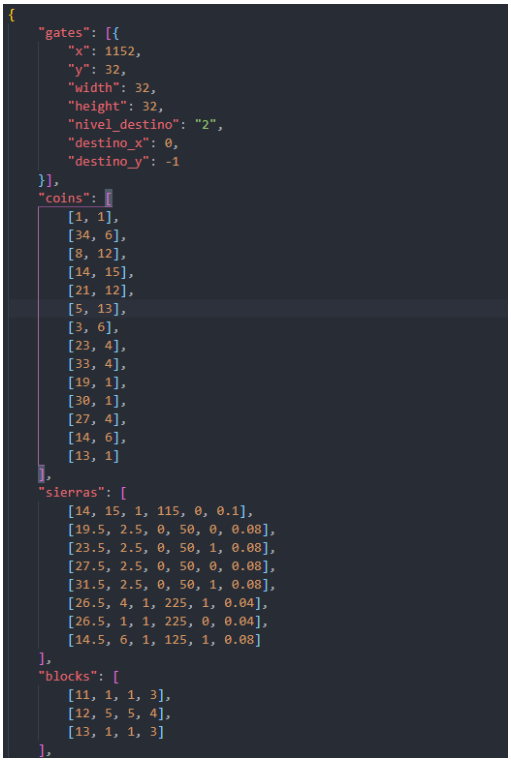
También tengo añadidos un EventListener para la captura de las distintas teclas que ejecutan la función keyDownUp vista anteriormente (la cual llama al método del objeto controller), un EventListener para inhabilitar los movimientos del scroll al pulsar las flechas y un EventListener para que al reescalar la página se ejecute la función resize.

```
//inicializar
var loader = new Loader();
var controller = new Controller();
var game = new Game();
var display = new Display(document.querySelector("canvas"));
var engine = new Engine(1000 / 30, render, update);

//cargar
display.buffer.canvas.height = game.world.height;
display.buffer.canvas.width = game.world.width;
display.buffer.imageSmoothingEnabled = false;

loader.rqJSON("json/nivel" + game.world.id_nivel + ".json", (nivel) => {
    game.world.cargarNivel(nivel);

    loader.rqTileImage("tiles/tiles.png", (imagen) => {
        loader.tile_set_image = imagen;
        resize();
        engine.start();
    })
})


window.addEventListener("keydown", keyDownUp);
window.addEventListener("keyup", keyDownUp);
window.addEventListener("keydown", function(e) {
    // space and arrow keys
    if ([32, 37, 38, 39, 40].indexOf(e.keyCode) > -1) {
        e.preventDefault();
    }
}, false)
window.addEventListener("resize", resize);
```

5.7. NIVELES.JSON

Aquí voy a enseñar como se diseña un nivel en el fichero json

Definimos los arrays de los objetos puertas, monedas, sierras y bloques



```
{
    "gates": [{
        "x": 1152,
        "y": 32,
        "width": 32,
        "height": 32,
        "nivel_destino": "2",
        "destino_x": 0,
        "destino_y": -1
    }],
    "coins": [
        [1, 1],
        [34, 6],
        [8, 12],
        [14, 15],
        [21, 12],
        [5, 13],
        [3, 6],
        [23, 4],
        [33, 4],
        [19, 1],
        [30, 1],
        [27, 4],
        [14, 6],
        [13, 1]
    ],
    "sierras": [
        [14, 15, 1, 115, 0, 0.1],
        [19.5, 2.5, 0, 50, 0, 0.08],
        [23.5, 2.5, 0, 50, 1, 0.08],
        [27.5, 2.5, 0, 50, 0, 0.08],
        [31.5, 2.5, 0, 50, 1, 0.08],
        [26.5, 4, 1, 225, 1, 0.04],
        [26.5, 1, 1, 225, 0, 0.04],
        [14.5, 6, 1, 125, 1, 0.08]
    ],
    "blocks": [
        [11, 1, 1, 3],
        [12, 5, 5, 4],
        [13, 1, 1, 3]
    ],
```

Definimos las variables del mapa como son el tamaño de los tiles, las filas y columnas del mapa, la posición del respawn de nuestro jugador, el mapa y el id del nivel definido

```
"tile_size": 32,
"columnas": 36,
"filas": 17,
"reaparicion_x": 1,
"reaparicion_y": 1,
"mapa": [
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    20, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
    20, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
    20, 1, 1, 1, 2, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 21,
    0, 1, 1, 1, 1, 0, 0, 0, 0, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    20, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
    0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 2, 1, 1, 0, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 0,
    19, 3, 3, 3, 3, 3, 3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 3, 3, 3, 3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
    0, 0, 0, 0, 0, 0, 21, 0, 0, 0, 0, 0, 0, 0, 21, 0, 0, 0, 0, 0, 0, 0, 0, 21, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0,
    0, 0, 0, 0, 0, 20, 21, 21, 0, 0, 0, 0, 20, 20, 20, 21, 0, 0, 0, 0, 0, 0, 20, 21, 20, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 0,
    0, 4, 4, 4, 4, 0, 20, 0, 1, 1, 1, 1, 0, 20, 20, 20, 1, 1, 1, 1, 1, 1, 0, 0, 0, 4, 4, 4, 4, 1, 1, 1, 2, 1, 1, 21,
    0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 20, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 2, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 0,
    20, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 20, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 21, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 0,
    20, 21, 21, 21, 0, 0, 0, 0, 0, 21, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
],
"id_nivel": "1"
```

Y con esto concluimos la explicación del desarrollo

6. POSIBLES MEJORAS

En este apartado voy a explicar las posibles mejoras que serían aplicables al código

6.1. OBJETIVO FINAL

Ahora mismo el objetivo del juego es conseguir monedas, pero no hay un objetivo definido ni una pantalla de victoria, me hubiera encantado implementarlo, pero por tiempo no existe victoria posible en el juego

6.2. CONTENIDO

Me gustaría meter más contenido como enemigos nuevos, tipo pinchos que se activen y desactiven con el tiempo, distintos niveles con formas distintas como un nivel en vertical, funcionalidades de movimiento como por ejemplo derrapar con las paredes y poder saltar si derrapamos en ellas etc.

6.3. RESPONSIVE

Ahora mismo el canvas simplemente se reescala pero tenía pensado y he intentado crear un elemento llamado viewport que haría como una cámara que seguiría al personaje e iría pintando el mapa a la que el personaje se mueve, por lo que cuando tengamos ventanas de navegadores de poca resolución o queramos hacer zoom por temas de visibilidad se cree este viewport para hacer así el juego “responsive”

6.4. COMPATIBILIDAD DISPOSITIVOS MOVILES

El juego, como tal, debería poder ser ejecutado en dispositivos móviles pero los controles no funcionarán, por lo que había pensado en un futuro añadir controles táctiles para estos dispositivos.

6.5. BASE DE DATOS CON PUNTUACIONES

He pensado también en una posible implementación de una base de datos de jugadores con puntuaciones basadas en el tiempo en completar el juego

6.6. PERSONALIZACIÓN

Poder personalizar el juego a tu gusto, como, por ejemplo, añadiendo niveles de dificultad donde se añadan más obstáculos en los niveles más difíciles mientras que en los más fáciles haya menos, aumentar o disminuir número de vidas según el nivel, etc.

También por ejemplo poder customizar tu personaje con distintas skin y skins de mapas

7. BIBLIOGRAFÍA

- Objetos JavaScript:
  https://www.w3schools.com/js/js_object_definition.asp

- Propiedades de los objetos JavaScript:
  https://www.w3schools.com/js/js_object_properties.asp

- Métodos JavaScript:
  https://www.w3schools.com/js/js_object_methods.asp

- Getters y Setters JavaScript:
  https://www.w3schools.com/js/js_object_accessors.asp

- Constructores JavaScript:
  https://www.w3schools.com/js/js_object_constructors.asp

- Prototypes JavaScript (Herencia):
  https://www.w3schools.com/js/js_object_prototypes.asp

- Dibujado de mapa:
  https://www.youtube.com/watch?v=XELyA6ECDLk

- Sistema de colisiones:
  https://www.youtube.com/watch?v=r-Y-N4cLd10

- Sistema de niveles:
  https://www.youtube.com/watch?v=96Q200cPFss

- Explicación de fixed time step loop:
  https://gamedev.stackexchange.com/questions/37187/semi-fixed-timestep-ported-to-javascript
  https://www.youtube.com/watch?v=34_sfONKDS4