Red Hood

Juego Web con jQuery y Canvas

Irene Viñas Ostos

Diseño de Interfaces Web

2º Desarrollo de Aplicaciones Web

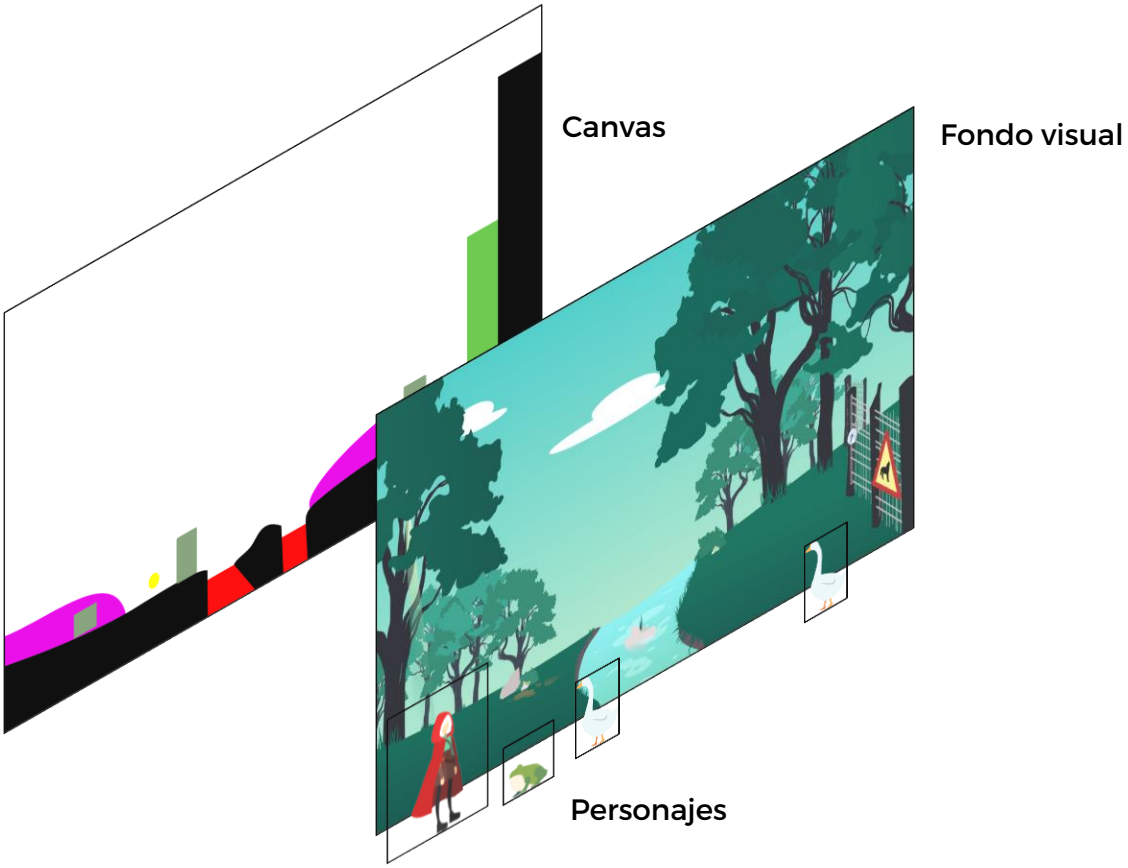
I.E.S. Hermanos Machado

Febrero 2020

Índice

0. Introducción

En este documento se explica el funcionamiento de un juego de navegador realizado con jQuery y Canvas. Se ha aprovechado la facilidad de encontrar puntos concretos relacionados con un objeto de Canvas y la versatilidad de jQuery para manipular el DOM de la web.



■ Colisiones (colisiones.js)

Las colisiones son calculadas a través de los colores que se han dibujado en el Canvas junto a una caja de contorno (*hitbox*) del personaje que se ha definido por puntos.

Si algún punto de la *hitbox* del personaje choca con algún color delimitado como "terreno", no podrá atravesarlo. De la misma manera no podrá pasar puertas cerradas.

Otros colores designan:

- ❖ Si es "vacío", que cuando el personaje lo toca es como si cayese.
- ❖ Si es "llave", "armas" u "objetos" y se pulsa cierta tecla cuando el personaje toca el color correspondiente, recibirá el objeto.

- Puerta abierta, si el personaje toca esta parte cambiará de mapa.
- Zona enemiga, designa en que puntos los enemigos aparecen y en caso de los enemigos voladores delimitan por donde deben volar.

Para comprobar cuando un personaje toca a un enemigo, se ha realizado un intervalo que dibuja la hitbox del enemigo en el Canvas con un color propio, si algún punto del personaje toca este color recibirá daño. En caso que lo haga un arma con el enemigo, el enemigo será quien lo reciba.

Físicas

Todo personaje terrestre es afectado por un intervalo que calcula la gravedad, de tal manera que aunque salten, volverán al suelo haciendo la trayectoria correspondiente. En el momento que el personaje tiene en sus pies un color de "terreno" dejará de ser afectado por la gravedad.

Clases

Para simplificar algunos aspectos del juego se ha creado diferentes clases que tienen sus propios atributos.

- **Dinámico**: Es la clase padre del resto de personajes que se mueven, tiene los métodos generales de colisiones y los atributos que coinciden como la capa DOM que tiene asignada.
- **Caperucita**: Es la clase del personaje principal, en él se define su movimiento y su forma de atacar. También tiene funciones que gestionan la vida del personaje y la imagen que se muestra.
- **Enemigo Terrestre**: Es la clase a la que pertenece los enemigos que están sobre el terreno y son afectados por la gravedad. Se definen funciones de movimiento propias.
- **Enemigo Volador**: Es la clase a la que pertenece los enemigos que "vuelan" y se mueven en cuatro direcciones.
- **Proyectil**: Es la clase a la que pertenece los diferentes tipos de proyectiles lanzados por el personaje principal.
- **Mapa**: Es la clase en la que se designa todas las características de cada nivel.

1. Animación de movimiento

Para completar el primer ejercicio que se pide, se realiza un cambio de imagen tipo gif cada vez que el personaje se mueve en sus diferentes direcciones. En el caso que el personaje esté sobre el suelo y no esté tocando ninguna tecla, esta imagen es remplazada por una estática.

Funciones que realizan el cambio de imagen (caperucita.js):

```
// Cambiar imagenes del personaje
animacion(img) {
    this.capa.css("background-image", "url('img/caperucita/" + img + ".gif')");
    this.img = img;
}
estatica(img) {
    this.capa.css("background-image", "url('img/caperucita/" + img + ".png')");
    this.img = img;
}
```

Funciones que controlan el cambio (caperucita.js):

```
// Movimiento
moverDerecha() {
    this.mirar = "right";
    this.direccion = "right";
    this.velocidadX = 7;
    this.animacion(teclasMovimiento.agacharse.on ? "down_right" : "run_right");
    this.capa.animate({ left: this.izquierda += this.velocidadX }, { duration: 10, queue: false }, "linear");
}
moverIzquierda() {
    this.mirar = "left";
    this.direccion = "left";
    this.velocidadX = -7;
    this.animacion(teclasMovimiento.agacharse.on ? "down_left" : "run_left");
    this.capa.animate({ left: this.izquierda += this.velocidadX }, { duration: 10, queue: false }, "linear");
}
agachate() {
    if (!teclasMovimiento.derecha.on && !teclasMovimiento.izquierda.on) {
        this.estatica("down_" + this.direccion);
    }
}
salta() {
    if (this.colisionaPorAbajo(10)) {
        this.velocidadY = -30;
        this.estatica("jump_" + this.direccion);
    }
}
```

2. Colisiones

Para resolver el segundo ejercicio, utilizo la clase “Dinámico” que define cuando personaje choca contra un obstáculo o una pared. En este juego, en vez habitaciones como tal, son mapas limitados por diferentes obstáculos.

Funciones principales (personajeDinamico.js):

```
// Colisiones
colisionaPorAbajo(px) {
    return colision(this.izquierda, this.arriba + px, this.contorno, "terreno") ||
        colision(this.izquierda, this.arriba + px, this.contorno, "puertaCerrada") ||
        colision(this.izquierda, this.arriba + px, this.contorno, "troncoBloqueando");
}
colisionaPorArriba(px) {
    return colision(this.izquierda, this.arriba - px, this.contorno, "terreno") ||
        colision(this.izquierda, this.arriba - px, this.contorno, "puertaCerrada") ||
        colision(this.izquierda, this.arriba - px, this.contorno, "troncoBloqueando");
}
colisionaPorDerecha(px) {
    return colision(this.izquierda + px, this.arriba, this.contorno, "terreno") ||
        colision(this.izquierda + px, this.arriba, this.contorno, "puertaCerrada") ||
        colision(this.izquierda + px, this.arriba, this.contorno, "troncoBloqueando");
}
colisionaPorIzquierda(px) {
    return colision(this.izquierda - px, this.arriba, this.contorno, "terreno") ||
        colision(this.izquierda - px, this.arriba, this.contorno, "puertaCerrada") ||
        colision(this.izquierda - px, this.arriba, this.contorno, "troncoBloqueando");
}
// Tocar diferentes objetos o partes del mapa
tocar(color, x, y) { return colision(this.izquierda + 10 + (x != undefined ? (this.direccion == "right" ? x : x * (-1)) : 0), this.arriba - 10 + (y !=
    undefined ? (this.direccion == "right" ? y : y * (-1)) : 0), this.contorno, color); }
```

En el caso de “tocar” comprueba diferentes colores para la colisión con personajes y objetos.

3. Enemigos

Para resolver la tercera parte, se ha creado una función que busca todos los puntos designado como “zona enemiga” y se crea un nuevo objeto tipo Enemigo en uno de sus puntos.

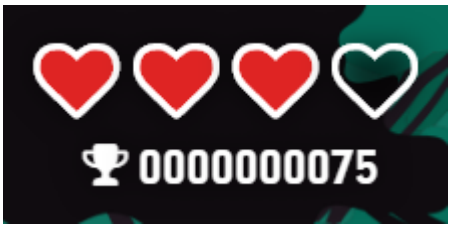
```
generaEnemigos() {
    let puntosPosibles = buscaColor("enemigos");
    for (let i = 0; i < this.enemigosGenerar.rana; i++) {
        let punto = Math.round(Math.random() * puntosPosibles.length);
        this.enemigos.push(new EnemigoTerrestre(puntosPosibles[punto], "rana", pRana, i, 7));
    }
    for (let i = 0; i < this.enemigosGenerar.ganso; i++) {
        let punto = Math.round(Math.random() * puntosPosibles.length);
        this.enemigos.push(new EnemigoTerrestre(puntosPosibles[punto], "ganso", pGanso, i, 7));
    }
    for (let i = 0; i < this.enemigosGenerar.murcielago; i++) {
        let punto = Math.round(Math.random() * puntosPosibles.length);
        this.enemigos.push(new EnemigoVolador(puntosPosibles[punto], "murcielago", pMurcielago, i));
    }
    if (this.enemigosGenerar.lobo) {
        this.enemigos.push(new EnemigoTerrestre({ x: 1200, y: 755 }, "lobo", pLobo, 150, 20));
    }
    if (this.enemigosGenerar.loboFeroz) {
        this.enemigos.push(new EnemigoTerrestre({ x: 1496, y: 715 }, "loboFeroz", pLoboFeroz, 230, 0));
    }
}
```

Con un número aleatorio, se elige un punto de esa zona aleatoria, haciendo así su aparición aleatoria pero dentro de un rango.

Cada mapa tiene una cantidad diferente de enemigos de diferentes tipos. Según aumenta el nivel, aumenta la cantidad de enemigos o la dificultad para derrotarlos.

En el caso que el personaje toque un color designado como “enemigo” perderá una vida y retrocederá para evitar perdiendo más vida. Cuando esto sucede, un mensaje indicando la pérdida de vida aparece con un efecto que cambia de tamaño y color:

Al mismo tiempo se actualiza el indicador de vida que hay en la esquina superior izquierda, junto a la puntuación que el jugador gana por realizar diferentes acciones:



4. Armas

Para completar el ejercicio cuatro, se han diseñado 3 tipos diferentes de armas que aparecen en diferentes niveles. Una de ellas es de contacto cercano, un hacha, que se encuentra en el primer nivel. Luego consigue una ballesta y una pulsera que lanza fuego.

Para que el disparo fuese más cómodo se ha incluido una guía que señala hacia dónde va a disparar y la flecha (o fuego) cambia de ángulo.

Se ha conseguido gracias a una función que calcula el ángulo y lo aplica al disparar con trigonometría:

```
function moverGuia(x, y) {
    let tx = x - personaje.izquierda;
    let ty = y - personaje.arriba;
    let atan = Math.atan2(ty, tx);
    let atanGrados = atan * (180 / Math.PI);
    let atanCalculado = atanGrados > 0.0 ? atanGrados : (360.0 + atanGrados);
    anguloLanzamiento = parseInt(atanCalculado);
    personaje.direccion = anguloLanzamiento >= 90 && anguloLanzamiento < 270 ? "left" : "right"

    $(".guia").css("transform", "rotate(" + anguloLanzamiento + "deg)");
}
```

Como cada disparo es un “proyectil”, cuando este colisiona con un enemigo o pared, desaparece. En el caso de chocar con un enemigo, este recibe daño y el jugador recibe 25 puntos.

5. Niveles

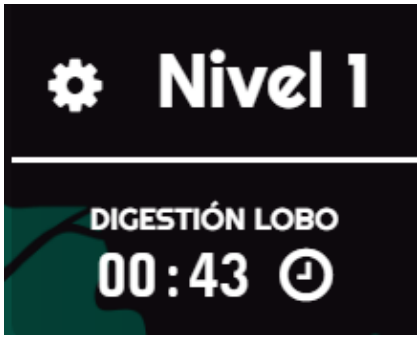
Como en el ejercicio cinco pide, se ha creado “puertas” que solo se pueden abrir con determinada “llave”. Con libertad creativa he realizado diferentes formas de crear ambas.

En el caso del nivel uno, la llave es un hacha que podrá cotar un árbol que está en medio del camino (puerta), una vez que el jugador realice esto, podrá pasar al nivel siguiente.

En el nivel dos, hay una llave escondida en un arbusto que abrirá el candado de la valla que impide el paso.

En el nivel tres, no hay llave ni puerta como tal, hay un sistema de palanca que al tirar de una cuerda, bajamos una rama que nos permite llegar a la puerta que pasa de nivel.

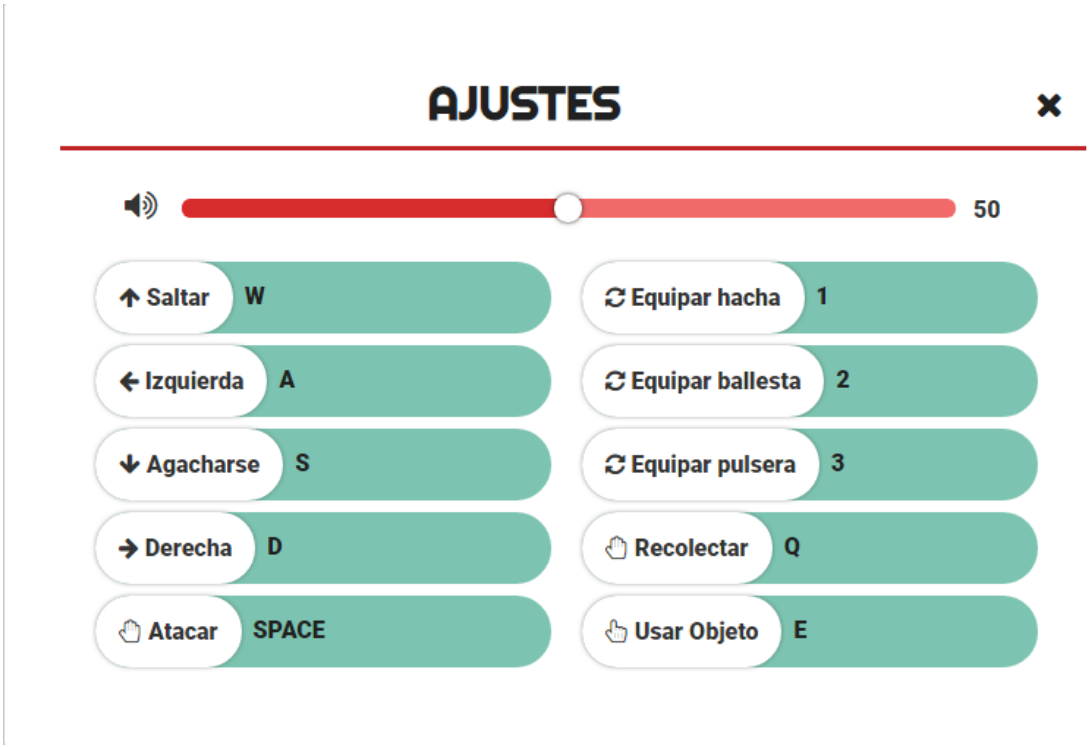
En todos los casos se usa la función “tocar” (personajeDinamico.js) y comprobar que se pulsa la tecla adecuada. Una vez que el personaje pasa al siguiente nivel, consigue 150 puntos y se actualizará el tiempo del nivel. En caso que este tiempo acabe sin terminar el nivel, este se reiniciará y el personaje perderá una vida.



6. jQuery UI

Se han implementado varias funciones de jQuery UI:

- Ajustes (dialog)



La ventana de ajustes es un diálogo modal personalizado que permite cambiar el volumen y cambiar los controles del personaje.

```
$("#dialog").dialog({
    appendTo: "body",
    autoOpen: false,
    modal: true,
    width: 700
});
```

- Volumen (slider)

Aparece en los ajustes (ajustes.js) y controla el volumen del audio que se reproduce de fondo.

```
$("#volumen").slider({
    value: 50,
    orientation: "horizontal",
    range: "min",
    animate: true,
    slide: function(event, ui) {
        $(".valorVolumen").text(ui.value);
        $("audio")[0].volume = ui.value / 100;
        if (ui.value == 0) {
            $("audio")[0].pause();
            $("#playMusic i:not('.fa-music')").switchClass("fa-pause", "fa-play");
        }
        if (ui.value > 0) {
            $("audio")[0].play();
            $("#playMusic i:not('.fa-music')").switchClass("fa-play", "fa-pause");
        }
    }
});
```

- Efectos (effect)

Cuando el personaje pierde un corazón, este salta en el sitio y se vacía gracias al efecto “shake”.

```
$("#vida i:nth-child(" + this.vida + "):not(.fa-trophy)")
    .effect("shake", { direction: "up", distance: 10, times: 2 }).addClass("perdida");
```

- Redimensionar (resizable)

Para tener todo visible en la pantalla, independientemente de su tamaño, se utiliza resizable.

```
// Hacer que el juego siempre tenga el tamaño correcto según la pantalla
$("#juego").resizable({
    minHeight: 150,
    minWidth: 200,
    aspectRatio: 16 / 9,
    containment: "body",
    stop: rescala()
});
```

7. Valoración

Para este proyecto he utilizado alrededor de 140 horas de trabajo, se podrían separar en varios apartados:

- Aprender tecnología Canvas → 7 horas
- Crear sistema de colisiones → 18 horas
- Diseño visual completo → 25 horas
- Programación general → 90 horas

Me he encontrado con dificultades para realizar las colisiones más fluidas posibles y, además, consumí muchas horas diseñando cada personaje y fondo del juego, así como su interfaz.

Aun así, me siento realizada por haber conseguido realizar un juego completo con tecnologías nuevas como Canvas. También me ha gustado recordar lo que aprendí de física y matemáticas para diferentes aspectos del juego, consolidando la teoría aprendida en algo práctico.

Sin embargo, si es cierto que algunas limitaciones puestas en el enunciado hacían alejarme de la idea principal que tenía del juego. Por ejemplo, reiniciar el nivel cuando se acaba un tiempo asignado a cada nivel en vez de tener un tiempo en todo el juego, que venía mejor con la temática.

Cabe señalar, que me habría gustado aprender más sobre los eventos de jQuery, ya que lo dado en clase solo tocaba la superficie y hay apartados muy interesantes como los eventos personalizados o hacer diferentes cosas dentro de un solo evento como *keydown*, que habría sido información de gran ayuda para este proyecto.