BASIC HTML

```
<!DOCTYPE html>…

    <title>26520266R</title>

    <style type="text/css">

        body {background-color: #a263dd; margin: auto; margin-top:3em;width: 80%;}

        a:link {color: #ff6600;}

        a:visited {color: #333333;}

        a:hover {color: #000066;}

        .myDiv {background: #1f4b02; border-color: #ffffff #ffffff #000000 #000000;
border-radius: 30px; box-shadow: 20px 20px 30px 10px #000000; border-width: 5px;
padding: 1em 1em 1em 1em; font-family: 'Handlee', cursive; font-size: 20pt; font-
weight: bold; text-align: justify; color: #41b6d3; margin-bottom: 2em; }

        .myParagraph {font-family: 'Handlee', cursive; font-weight: bolder; font-
size: 16pt; text-align: left; color: #e4d8d8f6; }

        .myUl {list-style-image:
url(http://acrata.act.uji.es/vj1217/bulletOrange.png); list-style-position: inside;
margin: 8px; }

        #specialLi {margin-top: 1em; margin-bottom: 1em; font-family: 'Handlee',
cursive; font-size: 14pt; text-decoration: underline; }

        #myweb {font-family: 'Gloria Hallelujah', cursive; font-size: 24pt; font-
weight: bolder; color:#ff6600}

    </style>

</head>

<body>

    <div class="myDiv">text</div>

    <ul class="myUl">

    <li id="specialLi" id="myweb"><a id="myweb"
href="http://teseo.act.uji.es/~al385775/DinoCangris/" title="Final Project: Cangris
Dev Division">Dino Cangris</a></li>

    <li id="specialLi"><a id="myweb"
href="http://teseo.act.uji.es/~al385776/Team18Project/" title="Bouncing
Ball">Bouncing Ball</a></li>

    </ul>

    <div class="myDiv">Bonus track: <a id="myweb"
href=http://teseo.act.uji.es/~al315320/Kadicks%20Resolution/

title="Play Kadick's Resolution game">this is another game developed with
Phaser</a>.text</div>

<div class="myDiv">texttt</div>

<div class="myDiv">blablabla</div>

<p class="myParagraph">brr</p>

<ul class="myUl"><li id="specialLi">I will learn …</li>
```

```
<li id="specialLi">Also, I ….</li>
</ul>
</body>
</html>
```

RADIO BUTTONS

```
let endScreen;
let textInGame;
let buttonInGame;
let playagainButton;
let textGameOver;


function start() {
    //Initial screen and game screen
    let initialScreen = document.getElementById("initialScreen");
    textInGame = document.getElementById("texto");
    buttonInGame = document.getElementById("boton");
    //End screen
    endScreen = document.getElementById("endScreen");
    playagainButton = document.getElementById("playagain");
    textGameOver = document.getElementById("textgameover");
    initialScreen.style.display = "none";
    playGame();
}
function playGame() {//When you play again, this screen was not displayed so we have
to activate it again
    gameScreen.style.display = "block";
    // Get the value of the radio button checked in the initial screen
    let radioButtons = document.getElementsByName("level");
    let choice, msg;
    for (const button of radioButtons) {
        if (button.checked) {
            choice = parseInt(button.value);
            break;
        }
    }
```

```
    // Set msg according to the level user chose
    switch (choice) {
        case 1:
            msg = "at the FIRST level.";
            break;
        case 2:
            msg = "at the SECOND level.";
            break;
        case 3:
            msg = "at the THIRD level.";;
            break;
        case 4:
            msg = "at the FOURTH level.";;
            break;
        default:
            // This should never be executed
            msg = "at NONE."
            break;
    }
    // Add contents to Game Screen
    textInGame.innerHTML = "You're PLAYING this game " + msg;
    let img = document.createElement("img");
    let att = document.createAttribute("src");
    att.value = "imgs/stop.jpg";
    img.setAttributeNode(att);
    att = document.createAttribute("width");
    att.value = "150px";
    img.setAttributeNode(att);
    att = document.createAttribute("height");
    att.value = "150px";
    img.setAttributeNode(att);
    att = document.createAttribute("onclick");
    att.value = "endGame()";
    img.setAttributeNode(att);
    buttonInGame.appendChild(img);
}
```

```
function endGame() {
    // Show end screen after 3 seconds
    window.setTimeout(showEndScreen, 3000);
}

function showEndScreen() {
    // Hide Game Screen
    let gameScreen = document.getElementById("gameScreen");
    gameScreen.style.display = "none";

    // I have added one div for the text and one for the button in the HTML doc: here
they are textGameOver and playAgainButton
    //The text:
    textGameOver.innerHTML = "The game is OVER!!!!"
    textGameOver.style.display = "block";

    //PLAY AGAIN button:
    let img = document.createElement("img");
    let att = document.createAttribute("src");
    att.value = "imgs/play_again.png";
    //Height and width
    img.setAttribute("width", "250px");
    img.setAttribute("height", "100px");
    //On click
    img.setAttributeNode(att);
    att = document.createAttribute("onclick");
    att.value = "restart()";
    img.setAttributeNode(att);
    //Margins
    att = document.createAttribute("style");
    att.value = "margin-top: 75%";
    img.setAttributeNode(att);
    att = document.createAttribute("margin-left");
    att.value = "auto";
    img.setAttributeNode(att);
    att = document.createAttribute("margin-right");
```

```
    att.value = "auto";

    img.setAttributeNode(att);

    //Display as a block

    att = document.createAttribute("display");

    att.value = "block";

    img.setAttributeNode(att);

    playagainButton.appendChild(img); //Added to the div

    //Show end screen

    endScreen.style.display = "block";

}

function restart(){

    endScreen.style.display = "none";   //Hide end screen and display the initial
screen to restart the game

    initialScreen.style.display = "block";

    buttonInGame.firstElementChild.remove(); //Remove the STOP image, if not, it will
appear many times!

    playagainButton.firstElementChild.remove(); //Same with the other image we added
for the play again button

    //We don't have to do it with the text because it is written in innerHTML

}
```

CSS Y HTML

```
#initialScreen {margin-top: 3em; margin-left: auto; margin-right: auto; width: 80%;}

#endScreen {width: 600px; height: 600px; background: url(../imgs/gameOver.png); text-
align: center; vertical-align: text-bottom; font-family: "Comic Sans MS", sans-serif;
font-weight: bolder; font-size: 24pt; font-variant: small-caps; display: none;}

#gameScreen {width: 70%;margin-top: 10em; margin-left: auto; margin-right: auto;}

#texto {font-family: "Comic Sans MS", sans-serif; font-weight: bold; font-size: 20pt;
background-color:
lightsalmon; color: crimson;}

#boton {margin-top: 5em; }

.radioGroup {display: block;
position: relative;padding-
left: 1em; margin-bottom:
15px; cursor: pointer; font-
size: 14pt;}

/* Resize the radio buttons
*/

input[type="radio"]
{transform: scale(1.3);}
```

```
<body>
    <div id="initialScreen">
        <div style="margin-bottom: 2em;">
            <span style="font-size: 18pt; font-weight: bolder; font-variant: small-caps;">Choose level:</span>
        </div>
        <label class="radioGroup">
            <input type="radio" name="level" value="1">
            This is for level 1
        </label>
        <label class="radioGroup">
            <input type="radio" checked="checked" name="level" value="2">
            This is for level 2
        </label>
        <label class="radioGroup">
            <input type="radio" name="level" value="3">
            This is for level 3
        </label>
        <label class="radioGroup">
            <input type="radio" name="level" value="4">
            This is for level 4
        </label>
        <div style="margin-top: 2em;">
            <img src="imgs/play.png" onclick="start()">
        </div>
    </div>
    <div id="gameScreen">
        <div id="texto"></div>
        <div id="boton"></div>
    </div>
    <div id="endScreen">
        <div id="textgameover"></div>
        <div id="playagain"></div>
    </div>
    <script src="js/main.js"></script>
</body>
```

PR1

```html
<!-- This is a comment in HTML -->
<h1>Web games with HTML</h1>
<p>We will need to learn:</p>
<ul>
    <li>HTML5</li>
    <li>CSS3</li>
    <li>JavaScript</li>
</ul>
<h1 class="resources">Web Links</h1>
<h1 class="resources books">Useful Books</h1>
<a href="https://www.juegos.com">Link to Juegos.com</a>
<div id="background">
    <h1>
        <input type="checkbox" id="back-gray" value="gray" onclick = "setGray()"> Gray background<br>
    </h1>
</div>
<div id="animacion">
    <img src="cheems.png" alt="cheems" id="cheems">
</div>

<canvas id="myCanvas" width="200" height="100" style="border:1px solid #d3d3d3;"></canvas>
<div>
    <input type="range" id="op_slider" onchange="setOpacity()">
</div>

<script>
    function setGray(){
        let color="white"; // value if not checked
        if (document.getElementById("back-gray").checked)
        color="gray"; // value if checked
        // get element with desired id
        const e = document.getElementById("background");
        // set the background color to that element
        e.style["background-color"] = color;
    }
        let can = document.getElementById("myCanvas");
        var ctx = can.getContext("2d");
        ctx.beginPath();
        ctx.arc(95,50,40,0,2*Math.PI);
        ctx.strokeStyle = "pink";
        ctx.lineWidth = 0;
        ctx.fillStyle = "pink";
        ctx.globalAlpha = 0.3;
        ctx.fill();
        ctx.stroke();

    let slider;
    setOpacity();

    function setOpacity(){
        slider = document.getElementById('op_slider');
        ctx.clearRect(0, 0, can.width, can.height);
        ctx.globalAlpha = slider.value * 0.01; //Valor del slider en %

        ctx.beginPath();
        let radio;
        radio = 0.4*Math.min(can.width, can.height);
        ctx.arc(95, 50, radio, 0, 2*Math.PI);
        ctx.fillStyle = 'pink';
        ctx.fill();
        ctx.stroke();
    }
</script>

</body>
</html>
```

```css
#cheems{
 margin-top: 5em;
 margin-left: 3em;
 position: absolute;
 animation-name: example;
 width: 30%;
 height: 30%;
 animation-duration: 4s;
 animation-iteration-count: infinite;
 animation-timing-function: linear;
}

@keyframes example {
    0%{left:0%;}
    50%{left:50%; transform: rotate(90deg);}
    100%{left:0%; transform: rotate(0deg);}
}
```

SCREEN CAR TRAVEL

```
class Car {
    constructor(x,y) {
        this.x = x;
        this.y = y;
    };
};
```

```
let myGame = {}; myGame.carIsMoving = false; myGame.gameIsOver = false; myGame.carImage;
myGame.car; myGame.canvas; myGame.context; myGame.timer;
// Entry point
window.onload = function () {
    myGame.init();
};
myGame.init = function () {
    // Hide all game layers and display the start screen
    myGame.hideAllScreens();
    document.getElementById("gamestartscreen").style.display = 'inline';
    // Get handler for game canvas and context
    myGame.canvas = document.getElementById("gamecanvas");
    myGame.context = myGame.canvas.getContext("2d");
};
myGame.hideAllScreens = function () {
    layers = document.getElementsByClassName("gamelayer");
    for ( let i = 0 ; i < layers.length ; i++ )
        layers[i].style.display = 'none';
};
myGame.showCanvas = function () {
    myGame.hideAllScreens();
    document.getElementById("gamecanvas").style.display = 'inline';
    myGame.play();
};
myGame.showEnd = function () {
    myGame.hideAllScreens();
    document.getElementById("endingscreen").style.display = 'inline';
    clearInterval(myGame.timer);
};
myGame.play = function () {
    myGame.car = new Car(0,270);
    myGame.carImage = new Image();
    myGame.carImage.src = 'images/car.png';

    myGame.carImage.onload = function () {
        myGame.context.drawImage(myGame.carImage, myGame.car.x, myGame.car.y);
    };
    myGame.mouseClick();
    // set interval to call gameloop in 50 FPS
    myGame.timer = setInterval(myGame.gameLoop, 1000/50);
};
myGame.mouseClick = function () {
    // Add Mouse Event Listener to canvas
    myGame.canvas.addEventListener("mousedown", function(e) {
        let mouseX = e.pageX - this.offsetLeft;
        let mouseY = e.pageY - this.offsetTop;
        if (mouseX >= myGame.car.x && mouseX <= myGame.car.x + myGame.carImage.naturalWidth &&
            mouseY >= myGame.car.y && mouseY <= myGame.car.y + myGame.carImage.naturalHeight)
            myGame.carIsMoving = !myGame.carIsMoving; });     };
```

```
myGame.gameLoop = function () {
    if ( !myGame.gameIsOver && myGame.carIsMoving ) {
        myGame.context.clearRect(myGame.car.x, myGame.car.y, myGame.carImage.naturalWidth, myGam
e.carImage.naturalHeight);
        myGame.car.x += 2;
        if (myGame.car.x >= myGame.canvas.width) {
            myGame.gameIsOver = true;
            myGame.showEnd(); }
        else
            myGame.context.drawImage(myGame.carImage, myGame.car.x, myGame.car.y); }
};
```

PR3: INFINITY SQUARE

```
const GAME_AREA_WIDTH = 800;
const GAME_AREA_HEIGHT = 500;
const SQUARE_SIZE = 40;
const SQUARE_COLOR = "#cc0000";
const SQUARE_SPEED_X = 5;
const SQUARE_SPEED_Y = 5;
let OBSTACLE_SPEED = 2;
let OBSTACLE_COLOR = "#187440";
const OBSTACLE_MIN_HEIGHT = 40;
const OBSTACLE_MAX_HEIGHT = GAME_AREA_HEIGHT - 100;
const OBSTACLE_WIDTH = 20;
const OBSTACLE_MIN_GAP = 65;
const OBSTACLE_MAX_GAP = GAME_AREA_HEIGHT - 100;
let PROBABILITY_OBSTACLE = 0.7;
const FRAME_OBSTACLE = 85;
const FPS = 30;
const CHRONO_MSG = "Time goes by...";
const RIGHTARROW_KEYCODE = 39;
const LEFTARROW_KEYCODE = 37;
const UPARROW_KEYCODE = 38;
const DOWNARROW_KEYCODE = 40;

class SquaredForm {
    constructor(x, y, width, height, color, img = null) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.color = color;
        this.speedX = 0;
        this.speedY = 0;
        this.img = img;
    }
    setSpeedX(speedX) {
        this.speedX = speedX; }
```

```
    setSpeedY(speedY) {
        this.speedY = speedY; }
    render(ctx) {
        if(this.img == null){
            ctx.fillStyle = this.color;
            ctx.fillRect(this.x, this.y, this.width, this.height); }
        else{
            ctx.drawImage(this.img, this.x, this.y); }
    }
    move() {
        this.x += this.speedX;
        this.y += this.speedY; }
    setIntoArea(endX, endY) {
        this.x = Math.min(Math.max(0, this.x), (endX - this.width));
        this.y = Math.min(Math.max(0, this.y), (endY - this.height)); }
    crashWith(obj) {
        // detect collision with the bounding box algorithm
        let myleft = this.x;
        let myright = this.x + this.width;
        let mytop = this.y;
        let mybottom = this.y + this.height;
        let otherleft = obj.x;
        let otherright = obj.x + obj.width;
        let othertop = obj.y;
        let otherbottom = obj.y + obj.height;
        let crash = true;
        if ((mybottom < othertop) || (mytop > otherbottom) || (myright < otherleft) ||
            (myleft > otherright)) {
            crash = false;
        }
        return crash;
    }
}
class GameArea {
    constructor(canvas, hero, enemies, obstacles) {
        this.canvas = canvas;
        this.hero = hero;
        this.enemies = enemies;
        this.obstacles = obstacles;
        this.context = null;
        this.interval = null;
        this.frameNumber = undefined; }
    initialise() {
        this.canvas.width = GAME_AREA_WIDTH;
        this.canvas.height = GAME_AREA_HEIGHT;
        this.context = this.canvas.getContext("2d");
        let theDiv = document.getElementById("gameplay");
        theDiv.appendChild(this.canvas);
        this.interval = setInterval(updateGame, 1000 / FPS);
```

```
        this.frameNumber = 0; }
    render() {
        for (const obstacle of this.obstacles) {obstacle.render(this.context); }
        for (const enemies of this.enemies) {enemies.render(this.context); }
        this.hero.render(this.context);
    }
    clear() {
     this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);}
    addObstacle(obstacle) {
        this.obstacles.push(obstacle);
    }
    addEnemy(enemy){
        this.enemies.push(enemy);
    }
    removeObstacle(i) {
        this.obstacles.splice(i, 1);
    }
    removeEnemy(i){
        this.enemies.splice(i,1); } }
    let spaceship = new Image();
    spaceship.src = "img/nave.png";
    let theSquare = new SquaredForm(0, GAME_AREA_HEIGHT / 2, SQUARE_SIZE, SQUARE_SIZE, SQUARE_CO
LOR, spaceship);
    let rightArrowPressed = false,
    leftArrowPressed = false,
    upArrowPressed = false,
    downArrowPressed = false;
    let seconds, timeout, theChrono;
    let continueGame = true;
    let gameArea = new GameArea(document.createElement("canvas"), theSquare, [], []);
function handlerOne(event) {
    switch (event.keyCode) {
        case RIGHTARROW_KEYCODE:
            if (!rightArrowPressed) {
                rightArrowPressed = true;
                theSquare.setSpeedX(SQUARE_SPEED_X); }
            break;
        case LEFTARROW_KEYCODE:
            if (!leftArrowPressed) {
                leftArrowPressed = true;
                theSquare.setSpeedX(-SQUARE_SPEED_X); }
            break;
        case UPARROW_KEYCODE:
            if (!upArrowPressed) {
                upArrowPressed = true;
                theSquare.setSpeedY(-SQUARE_SPEED_Y); }
            break;
        case DOWNARROW_KEYCODE:
            if (!downArrowPressed) {
```

```
                downArrowPressed = true;
                theSquare.setSpeedY(SQUARE_SPEED_Y); }
            break;
        default:
            break; }}
function handlerTwo(event) {
    switch (event.keyCode) {
        case RIGHTARROW_KEYCODE:
            rightArrowPressed = false;
            theSquare.setSpeedX(0);
            break;
        case LEFTARROW_KEYCODE:
            leftArrowPressed = false;
            theSquare.setSpeedX(0);
            break;
        case UPARROW_KEYCODE:
            upArrowPressed = false;
            theSquare.setSpeedY(0);
            break;
        case DOWNARROW_KEYCODE:
            downArrowPressed = false;
            theSquare.setSpeedY(0);
            break;
        default:
            break; }
}
function startGame() {
    let levels = document.getElementById('levels');
    levels.style.display = 'none';

    let radioButtons = document.getElementsByName("level");
    let choice;
    for (const button of radioButtons) {
        if (button.checked) {
            choice = parseInt(button.value);
            break; }}
    if(choice == 'difficult'){
        OBSTACLE_SPEED = 10;
        PROBABILITY_OBSTACLE = 0.9;
        OBSTACLE_COLOR = "#197460";}
    gameArea.initialise();
    gameArea.render();
    window.document.addEventListener("keydown", handlerOne);
    window.document.addEventListener("keyup", handlerTwo);
    seconds = 0;
    timeout = window.setTimeout(updateChrono, 1000);
    theChrono = document.getElementById("chrono");
}
```

```
let counter = 1;
function updateGame() {
    // Check collision for ending game
    let collision = false;
    for (let i = 0; i < gameArea.obstacles.length; i++) {
        if (theSquare.crashWith(gameArea.obstacles[i])) {
            collision = true;
            break;        }}
    for (let j = 0; j < gameArea.enemies.length; j++){
        if (theSquare.crashWith(gameArea.enemies[j])) {
            gameArea.removeEnemy(j);
            if (seconds >= 15) {seconds -= 15;}
            else{seconds = 0;}
            break}}
    if (collision) {
        let lifes = document.getElementById("lifes");
        if (counter<4) {
      lifes.removeChild(document.getElementById("life"+String(counter)));
            theSquare.x = 0;
            theSquare.y = GAME_AREA_HEIGHT / 2;
            counter++;
        }
        else{endGame();}
    }
    else {
        // Increase count of frames
        gameArea.frameNumber += 1;
        // Let's see if new obstacles must be created
        if (gameArea.frameNumber >= FRAME_OBSTACLE)
            gameArea.frameNumber = 1;
        // First: check if the given number of frames has passed
        if (gameArea.frameNumber == 1) {
            // Enemy creation
            let random = Math.random();
                formRival = new SquaredForm(GAME_AREA_WIDTH, Math.floor(Math.random() * GAME_ARE
        A_HEIGHT), SQUARE_SIZE/2,
            SQUARE_SIZE/2, "#000000");
            formRival.setSpeedX(random*-OBSTACLE_SPEED * 3);
            formRival.setSpeedY(random*-OBSTACLE_SPEED * 3);
            gameArea.addEnemy(formRival);
            // Obstacle creation
            let chance = Math.random();
          if (chance < PROBABILITY_OBSTACLE) {
            let height = Math.floor(Math.random() * (OBSTACLE_MAX_HEIGHT - OBSTACLE_MIN_HEIGHT +
                    1) + OBSTACLE_MIN_HEIGHT);
            let gap = Math.floor(Math.random() * (OBSTACLE_MAX_GAP - OBSTACLE_MIN_GAP + 1) +
                    OBSTACLE_MIN_GAP);
            let form = new SquaredForm(gameArea.canvas.width, 0, OBSTACLE_WIDTH, height, OBSTACLE_COLOR);
                form.setSpeedX(-OBSTACLE_SPEED);
```

```
                gameArea.addObstacle(form);
                // The obstacle at the bottom only is created if there is enough room
                if ((height + gap + OBSTACLE_MIN_HEIGHT) <= gameArea.canvas.height) {
                    form = new SquaredForm(gameArea.canvas.width, height + gap, OBSTACLE_WIDTH,
                        gameArea.canvas.height - height - gap, OBSTACLE_COLOR);
                    form.setSpeedX(-OBSTACLE_SPEED);
                    gameArea.addObstacle(form); }}}
    // Move obstacles and delete the ones that goes outside the canvas
        for (let i = gameArea.obstacles.length - 1; i >= 0; i--) {
            gameArea.obstacles[i].move();
            if (gameArea.obstacles[i].x + OBSTACLE_WIDTH <= 0) {
                gameArea.removeObstacle(i); }}
   // Move enemies and delete the ones that go outside the canvas
        for (let j = gameArea.enemies.length - 1; j >= 0; j--){
            gameArea.enemies[j].move();
            if (gameArea.enemies[j].x + SQUARE_SIZE/2 <= 0) {
                gameArea.removeEnemy(j); }
            if (gameArea.enemies[j].y + SQUARE_SIZE/2 <= 0 || gameArea.enemies[j].y + SQUARE_SIZ
E/2 >= GAME_AREA_HEIGHT){ gameArea.enemies[j].speedY *= -1; }}
        // Move our hero
        theSquare.move();
        // Our hero can't go outside the canvas
    theSquare.setIntoArea(gameArea.canvas.width, gameArea.canvas.height);
        gameArea.clear();
        gameArea.render();}}

function updateChrono() {
    if (continueGame) {
        seconds++;
        let minutes = Math.floor(seconds / 60);
        let secondsToShow = seconds % 60;
        theChrono.innerHTML = CHRONO_MSG + " " + String(minutes).padStart(2, "0") + ":" +
            String(secondsToShow).padStart(2, "0");
        timeout = window.setTimeout(updateChrono, 1000); } }
function endGame() {
    continueGame = false;
    clearInterval(gameArea.interval);
    window.document.removeEventListener("keydown", handlerOne);
    window.document.removeEventListener("keyup", handlerTwo);
    // Hide Game Screen
    let gameScreen = document.getElementById("initialScreen");
    gameScreen.style.display = "none";
    // GAME OVER
    let endScreen = document.getElementById("endScreen");
    let header = document.createElement("header");
    let msg = document.createTextNode("GAME OVER");
    header.append(msg);
    let record = document.createTextNode("You have achived " + seconds + " seconds. Congratulati
ons!!!");
```

```
<header>
    Drive the square to avoid obstacles!
</header>
<div id='levels'>
    <div style="margin-bottom: 2em;">
        <span style="font-size: 18pt; font-weight: bolder; font-variant: small-caps;">Choose
    </div>
    <label class="radioGroup">
        <input type="radio" name="level" value="easy">
        This is for level easy
    </label>
    <label class="radioGroup">
        <input type="radio" checked="checked" name="level" value="difficult">
        This is for level difficult
    </label>

    <div style="margin-top: 2em;">
        <img src="img/play.png" onclick="startGame()">
    </div>
</div>

<div id='main'>

    <p id="chrono"></p>
    <div id="gameplay"></div>
    <div id="lifes">
        <img id="life3" src="img/nave.png" alt="life 3">
        <img id="life2" src="img/nave.png" alt="life 2">
        <img id="life1" src="img/nave.png" alt="life 1">
    </div>

</div>
<footer>
    Use the arrow keys to drive the square
</footer>
</div>
```

```
    endScreen.appendChild(header);
    endScreen.appendChild(record);
    // Show End Screen
    endScreen.style.display = "inline-block";
}
```

PR4: LETTERS

```
const GAME_AREA_WIDTH = 400;
const GAME_AREA_HEIGHT = 400;
const BG_COLOR = "#4488AA";
const HITS_MIN = 5; //Si se alcanza este numero se acaba el juego
let letters;
let scoreText;
let numHits;
let musica;
// game instance
let game = new Phaser.Game(GAME_AREA_WIDTH, GAME_AREA_HEIGHT,
    Phaser.CANVAS, "game");
// state's phases
let mainState = {
    preload: preloadAssets,
    create: initGame,
    update: updateGame};
//--methods for each phase of the state--
function preloadAssets() {
    game.load.image("logo", "assets/imgs/uji.png");
    game.load.image("play", "assets/imgs/button_play.png");
    game.load.image("stop", "assets/imgs/button_stop.png");
    game.load.audio('gameMusic', 'assets/snds/GameTheme.mp3');}
function initGame() {
    game.stage.backgroundColor = BG_COLOR;
    logoImg = game.add.image(0, 0, "logo");
    logoImg.scale.setTo(0.2);
    playImg = game.add.button(0, 180, 'play', buttonPlay);
    stopImg = game.add.button(0, 180, 'stop', buttonPlay);
    stopImg.visible = false;
    musica = game.add.audio('gameMusic');
    game.sound.setDecodedCallback(musica, startMusic, this);
    game.input.keyboard.onDownCallback = getKeyboardInput; //Registra tecla pulsada
    letters = game.add.group(); //Grupo para letras
    letters.inputEnableChildren = true; //Permite que se pueda hacer clic en ellas
    letters.onChildInputDown.add(processLetter); //Evento para detectar clic en las letras
    game.input.onDown.add(checkDistance); //Evento para detectar clic (en el juego en general)
    //Contador de letras clickeadas
    numHits = 0;
    scoreText = game.add.text(
        15, // x
        logoImg.y+logoImg.height, // y
```

```
        'Hits: '+numHits, // text
        {fontSize: '32px', fill: '#000'});}
function updateGame() {
    moveLogo();
    shakeLetters();
    updateScore();
    checkGameOver();}
//--Eventos--Teclado
function startMusic(){
    musica.loop = true;
    musica.play();}
function getKeyboardInput(e) {
    if (e.keyCode >= Phaser.Keyboard.A
    && e.keyCode <= Phaser.Keyboard.Z) {
        let a = game.add.text(Math.random() * game.width,
        Math.random() * game.height, e.key,
        {fontSize: '40px', fill: '#FA2'},
        letters); // group to add to
        a.anchor.setTo(0.5, 0.5); } }
//Clic letras: si haces clic en una, la borra
function processLetter(item, pointer) {
    numHits++;
    item.destroy(); // frees up memory
    // kill() removes it from display list,
    // but not from the group}
//Clic normal: borrará todas las letras si se hace clic en el espacio de juego
function checkDistance() {
    const DIST_THRESH = 50;
    //X e Y donde el usuario hace clic
    let xPointer = game.input.x;
    let yPointer = game.input.y;
    for (const letter of letters.children) {
    let x = letter.x;
    let y = letter.y;
    let d = Math.sqrt(Math.pow(x-xPointer, 2) + Math.pow(y-yPointer, 2));
    //Distancia euclidea: raiz((x2-x1)^2+(y2-y1)^2)
    if (d <= DIST_THRESH)
    processLetter(letter); }}
function buttonPlay(){
    if (playImg.visible) {
        playImg.visible = false;
        stopImg.visible = true;}
    else {stopImg.visible = false; playImg.visible = true; }}
//--Funciones de updateGame--
function moveLogo(){
    logoImg.x += 1;
    //Para que no desaparezca a la derecha
    if(logoImg.x == game.width - logoImg.width) logoImg.x = 0;}
function shakeLetters() {
```

```
    for (const child of letters.children) {
    child.x += Math.random() * 2 - 1;
    child.y += Math.random() * 2 - 1;
    child.angle += Math.random() * 10 - 5; }}
function updateScore(){
    scoreText.text = 'Hits:' + numHits;}
function checkGameOver(){
    if(numHits >= HITS_MIN){
        musica.stop();
        game.state.start('over'); }}
// Entry point
window.onload = startGame;
function startGame() {
    game.state.add('main', mainState);
    game.state.add('over', overState);
    game.state.start('main');    }
```

OVER STATE

```
let overState = {
    preload: preloadTexture,
    create: createOver,
    update: updateAnimation };
var capguy;
var capguyFlip;
function preloadTexture(){
    game.load.atlasJSONHash('cityscene', 'cityscene.png', 'cityscene.json'); //Assets animacion}
function createOver(){
    game.stage.backgroundColor = "#FA2";
    scoreText = game.add.text(15, //x 0, //y'Game complete!\nYou got 5 hits\nClick to play again
', {fontSize: '32px', fill: '#4488AA'});
    scoreText.inputEnabled = true;
    scoreText.events.onInputDown.add(replayGame);
    //Animacion chico
    capguy = game.add.sprite(0, 180, 'cityscene', 'capguy/walk/0001');
    capguy.scale.setTo(0.5,0.5);
    capguy.animations.add('walk', Phaser.Animation.generateFrameNames('capguy/walk/', 1, 8, '',
4), 10, true, false);
    capguy.animations.play('walk');
    //Animación chico invertido
    capguyFlip = game.add.sprite(380, 180, 'cityscene', 'capguy/walk/0001');
    capguyFlip.scale.setTo(-0.5,0.5);
    capguyFlip.animations.add('walk2', Phaser.Animation.generateFrameNames('capguy/walk/', 1, 8,
 '', 4), 10, true, false);
    capguyFlip.animations.play('walk2');
}
function updateAnimation(){
    /*capguy.x += 3;
    if(capguy.x > 800)
    {capguy.x = -50; }*/
}
```

```
function replayGame(){
    game.state.start('main');
}
```

PR6: A SIMPLE PLAATFORM GAME

PLAY.JS

```
const ...
let levelsData = ['assets/levels/level01.json', 'assets/levels/level02.json'];
let playState = {preload: loadPlayAssets,: create: createLevel, update: updateLevel,};
let hudGroup, healthBar, healthValue, healthTween, hudTime;
let remainingTime;
let levelConfig;
let platforms, ground;
let enemies = [];
let player, cursors;
let toRight = false;
let firstAids, stars;
let totalNumOfStars;
let soundDamaged, soundCollectStar, soundGetAid, soundHitEnemy, soundOutOfTime, soundLevelPassed
let playerPlatform;
let exit;
let timerClock;
let exitingLevel;
class Enemy {
    constructor(spritesheet, tween, plat, right, limit, hits, isPatrolling = true) {
        this.sprite = spritesheet;
        this.flash = tween;
        this.platform = plat;
        this.faceright = right;
        this.stepLimit = limit;
        this.origX = spritesheet.x;
        this.hitsToBeKilled = hits;
        this.isPatrolling = isPatrolling;
    }
    getIsPatrolling() {
        return this.isPatrolling;
    }
    setIsPatrolling(patrols) {
        this.isPatrolling = patrols;
    }
    getHitsToBeKilled() {
        return this.hitsToBeKilled;
    }
    setHitsToBeKilled(hits) {
        this.hitsToBeKilled = hits;
    }
    patrol() {
        this.sprite.body.velocity.x = ENEMY_VELOCITY * this.sprite.scale.x;
```

```
        this.sprite.animations.play('run');
        if (this.faceright) {
            if (this.sprite.body.x >= this.stepLimit) {
                this.sprite.body.x = this.stepLimit - ENEMY_X_OFFSET;
                this.sprite.scale.x *= -1;
            } else if (this.sprite.body.x <= this.origX) {
                this.sprite.body.x = this.origX + ENEMY_X_OFFSET;
                this.sprite.scale.x *= -1; }
        } else {
            if (this.sprite.body.x <= this.stepLimit) {
                this.sprite.body.x = this.stepLimit + ENEMY_X_OFFSET;
                this.sprite.scale.x *= -1;
            } else if (this.sprite.body.x >= this.origX) {
                this.sprite.body.x = this.origX - ENEMY_X_OFFSET;
                this.sprite.scale.x *= -1; }
        }
    }
    attack(thePlayer) {
        let endOfPlatform, playerAtRight;
        // Enemy must face the player to attack
        if (this.sprite.body.x < thePlayer.body.x) {
            this.sprite.scale.x = 1;
            playerAtRight = true; } else {
            this.sprite.scale.x = -1;
            playerAtRight = false; }
        // Play attack animation and change velocity
        this.sprite.body.velocity.x = Math.trunc(ENEMY_VELOCITY * 1.4) * this.sprite.scale.x;
        this.sprite.animations.play('swing');
        // Set the right limits for the enemy's movement (attack and patrolling)
        if (this.faceright) {
            if (playerAtRight && thePlayer.body.x > this.stepLimit) {
                endOfPlatform = this.platform.x + this.platform.width;
this.stepLimit = Math.min(Math.min(endOfPlatform, game.world.width), thePlayer.body.x)-ENEMY_X_OFFSET;}
            if (!playerAtRight && thePlayer.body.x < this.origX) {
                this.origX = Math.max(Math.max(0, this.platform.x), thePlayer.body.x) +
                    ENEMY_X_OFFSET; }   } else {
            if (playerAtRight && thePlayer.body.x > this.origX) {
                endOfPlatform = this.platform.x + this.platform.width;
this.origX = Math.min(Math.min(endOfPlatform, game.world.width), thePlayer.body.x)-ENEMY_X_OFFSET; }
            if (!playerAtRight && thePlayer.body.x < this.stepLimit) {
                this.stepLimit = Math.max(Math.max(0, this.platform.x), thePlayer.body.x) +
                    ENEMY_X_OFFSET; }      }
        // Check that the enemy is not out of bounds. If out of bounds set velocity to 0
        if (this.faceright && (this.sprite.body.x >= this.stepLimit ||
                this.sprite.body.x <= this.origX))
            this.sprite.body.velocity.x = 0;
        if (!this.faceright && (this.sprite.body.x >= this.origX ||
                this.sprite.body.x <= this.stepLimit))
            this.sprite.body.velocity.x = 0; }}
```

```
function loadPlayAssets() {
    loadSprites();
    loadImages();
    loadSounds();
    loadLevel(levelToPlay);}
function loadSprites() {
    game.load.spritesheet('collector', 'assets/imgs/dude.png', 32, 48);
    game.load.spritesheet('enemy', 'assets/imgs/enemySprite.png', 55, 53, 15);}
function loadImages() {
    game.load.image('bgGame', 'assets/imgs/bgPlay.jpg');
    game.load.image('exit', 'assets/imgs/exit.png');
    game.load.image('ground', 'assets/imgs/platform.png');
    game.load.image('star', 'assets/imgs/star.png');
    game.load.image('aid', 'assets/imgs/firstaid.png');
    game.load.image('healthHolder', 'assets/imgs/health_holder.png');
    game.load.image('healthBar', 'assets/imgs/health_bar.png');
    game.load.image('heart', 'assets/imgs/heart.png');}
function loadSounds() {
    game.load.audio('damaged', 'assets/snds/hurt1.wav');
    game.load.audio('collectstar', 'assets/snds/cling.wav');
    game.load.audio('getaid', 'assets/snds/wooo.wav');
    game.load.audio('hitenemy', 'assets/snds/snare.wav');
    game.load.audio('outoftime', 'assets/snds/klaxon4-dry.wav');
    game.load.audio('levelpassed', 'assets/snds/success.wav');}
function loadLevel(level) {
    game.load.text('level', levelsData[level - 1], true);}
function createLevel() {
    exitingLevel = false;
    // Set World bounds (same size as the image background in this case)
    game.world.setBounds(0, 0, WORLD_WIDTH, WORLD_HEIGHT);

    // Background
    let bg = game.add.tileSprite(0, 0, game.world.width, game.world.height, 'bgGame');
    // Smooth scrolling of the background in both X and Y axis
    bg.scrollFactorX = 0.7;
    bg.scrollFactorY = 0.7;
    // Collide with this image to exit level
    exit = game.add.sprite(game.world.width - 100, game.world.height - 64, 'exit');
    game.physics.arcade.enable(exit);
    exit.anchor.setTo(0, 1);
    exit.body.setSize(88, 58, 20, 33);
    // Create sounds
    createSounds();
    // Create groups with a pool of objects
    createAids();
    createStars();
    totalNumOfStars = 0;
    // Get level data from JSON
    levelConfig = JSON.parse(game.cache.getText('level'));
```

```
    platforms = game.add.group();
    platforms.enableBody = true;
    // Create ground and platforms (with enemies, stars and aids) according to JSON data
    // Be aware that enemies ara not in a group. Each enemy is an instance and is stored in the
array enemies
    createGround();
    createPlatforms();
    // Now, set time and create the HUD
    remainingTime = secondsToGo;
    createHUD();
    //Create player. Initial position according to JSON data
    player = game.add.sprite(levelConfig.collectorStart.x, game.world.height -
        levelConfig.collectorStart.y, 'collector');
    player.anchor.setTo(0.5, 0.5);
    game.physics.arcade.enable(player);
    //  Player physics properties. Give the little guy a slight bounce.
    player.body.bounce.y = 0.2;
    player.body.gravity.y = BODY_GRAVITY;
    player.body.collideWorldBounds = true;
    // Camera follows the player inside the world
    game.camera.follow(player);
    //  Our two animations, walking left and right.
    player.animations.add('left', [0, 1, 2, 3], 10, true);
    player.animations.add('right', [5, 6, 7, 8], 10, true);
    //  Our controls.
    cursors = game.input.keyboard.createCursorKeys();
    // Update elapsed time each second
    timerClock = game.time.events.loop(Phaser.Timer.SECOND, updateTime, this);
}
function createSounds() {
    soundDamaged = game.add.audio('damaged');
    soundCollectStar = game.add.audio('collectstar');
    soundGetAid = game.add.audio('getaid');
    soundHitEnemy = game.add.audio('hitenemy');
    soundOutOfTime = game.add.audio('outoftime');
    soundLevelPassed = game.add.audio('levelpassed');}
function createAids() {
    firstAids = game.add.group();
    firstAids.enableBody = true;
    firstAids.createMultiple(MAX_AIDS, 'aid');
    firstAids.forEach(setupItem, this);}
function createStars() {
    // similar to the code above
    stars = game.add.group();
    stars.enableBody = true;
    stars.createMultiple(MAX_STARS, 'star');
    stars.forEach(setupItem, this);}
function setupItem(item) {
    item.anchor.setTo(0.5, 0.5);
```

```
    item.body.gravity.y = BODY_GRAVITY;}
function createGround() {
    ground = platforms.create(0, game.world.height - 64, 'ground');
    ground.scale.setTo(2.75, 2); // 400x32 ---> 1100x64
    ground.body.immovable = true;
    for (let i = 0, max = levelConfig.ground.enemies.length; i < max; i++)
        setupEnemy(levelConfig.ground.enemies[i], ground);
    for (let i = 0, max = levelConfig.ground.aids.length; i < max; i++)
        setupAid(levelConfig.ground.aids[i], ground.y);
    for (let i = 0, max = levelConfig.ground.stars.length; i < max; i++)
        setupStar(levelConfig.ground.stars[i], ground.y);}
function createPlatforms() {
    levelConfig.platformData.forEach(createPlatform, this);}
let n = 0;
function createPlatform(element) {
    platform = platforms.create(levelConfig.platformData[n].x, levelConfig.platformData[n].y, 'g
round');
    platform.scale.setTo(1, 1);
    platform.body.immovable = true;
    for (let i = 0, max = levelConfig.platformData[n].enemies.length; i < max; i++)
        setupEnemy(levelConfig.platformData[n].enemies[i], platform);
    for (let i = 0, max = levelConfig.platformData[n].aids.length; i < max; i++)
        setupAid(levelConfig.platformData[n].aids[i], platform.y);
    for (let i = 0, max = levelConfig.platformData[n].stars.length; i < max; i++)
        setupStar(levelConfig.platformData[n].stars[i], platform.y);
    // similar to the code of createGround
    n++;}
function setupEnemy(enemy, plat) {
    let isRight, limit;
    let theEnemy = game.add.sprite(enemy.x, plat.y - ENEMY_Y_OFFSET, 'enemy');
    theEnemy.anchor.setTo(0.5, 0.5);
    if (enemy.right === 0) {
        theEnemy.scale.x = -1;
        isRight = false;
        limit = Math.max(Math.max(0, plat.x) + ENEMY_X_OFFSET, enemy.x - ENEMY_STEP_LIMIT);
    } else {
        isRight = true;
        limit = Math.min(Math.min(plat.x + plat.width, game.world.width) - ENEMY_X_OFFSET,
            enemy.x + ENEMY_STEP_LIMIT);}
    let flash = game.add.tween(theEnemy).to({alpha: 0.0}, 50, Phaser.Easing.Bounce.Out)
        .to({alpha: 0.8}, 50, Phaser.Easing.Bounce.Out)
        .to({alpha: 1.0}, 50, Phaser.Easing.Circular.Out);
    game.physics.arcade.enable(theEnemy);
    theEnemy.body.immovable = true;
    theEnemy.body.collideWorldBounds = true;
    theEnemy.body.setSize(41, 43, 3, 10);
    theEnemy.animations.add('swing', [0, 1, 2, 3, 4, 5, 6, 7], 10, true);
    theEnemy.animations.add('run', [8, 9, 10, 11, 12, 13, 14], 10, true);
    let newEnemy = new Enemy(theEnemy, flash, plat, isRight, limit, jumpsToKill);
```

```
    enemies.push(newEnemy); }
function setupAid(aid, floorY) {
    let item = firstAids.getFirstExists(false);
    if (item)
        item.reset(aid.x, floorY - AID_STAR_Y_OFFSET); }
function setupStar(star, floorY) {
    let item = stars.getFirstExists(false);
    if (item) {
        item.reset(star.x, floorY - AID_STAR_Y_OFFSET);
        totalNumOfStars += 1; }}
function createHUD() {
    hudGroup = game.add.group();
    hudGroup.create(5, 5, 'heart');
    hudGroup.create(50, 5, 'healthHolder');
    healthBar = hudGroup.create(50, 5, 'healthBar');
    hudTime = game.add.text(295, 5, setRemainingTime(remainingTime), {
        font: 'bold 14pt Sniglet',
        fill: '#b60404'});
    hudGroup.add(hudTime);
    hudGroup.fixedToCamera = true;
    healthValue = MAX_HEALTH;}
function updateLevel() {
    let dist;
    //  The player collide with the platforms. Got it!
 let hitPlatform = game.physics.arcade.collide(player, platforms, playerInPlatform, null, this);
    // Stars and first-aid boxes collide with platforms
    game.physics.arcade.collide(stars, platforms);
    game.physics.arcade.collide(firstAids, platforms);
    // Check if player overlaps with any of the stars or first aids
    game.physics.arcade.overlap(player, stars, collectStar, null, this);
    game.physics.arcade.overlap(player, firstAids, getFirstAid, null, this);
    // Test collisions with enemies
    for (let i = enemies.length - 1; i >= 0; i--) {
        if (enemies[i].platform === playerPlatform) {
            dist = Phaser.Math.distance(enemies[i].sprite.body.x, enemies[i].sprite.body.y,
                player.body.x, player.body.y);
            if (Math.round(dist) <= ENEMY_DISTANCE_ATTACK)
                enemies[i].setIsPatrolling(false);
            else
                enemies[i].setIsPatrolling(true);
        } else
            enemies[i].setIsPatrolling(true);
        if (enemies[i].getIsPatrolling())
            enemies[i].patrol();
        else
            enemies[i].attack(player);
        if (game.physics.arcade.collide(player, enemies[i].sprite))
            playerVsEnemy(player, enemies[i].sprite, enemies[i], i); }
   //  Reset the players velocity (movement)
```

```
    player.body.velocity.x = 0;
    if (cursors.left.isDown) {
        //  Move to the left
        player.body.velocity.x = -PLAYER_VELOCITY;
        player.animations.play('left');
        toRight = false;
    } else if (cursors.right.isDown) {
        //  Move to the right
        player.body.velocity.x = PLAYER_VELOCITY;
        player.animations.play('right');
        toRight = true;
    } else {
        //  Stand still
        stopPlayer();}
    // Allow the player to jump if touching the ground.
    if (cursors.up.isDown && player.body.touching.down && hitPlatform) {
        player.body.velocity.y = -PLAYER_VELOCITY * 2;
        playerPlatform = undefined; }
    // Check if player exits level and the game is over
    if (!exitingLevel)
        game.physics.arcade.overlap(player, exit, endLevel, null, this); }
function playerInPlatform(player, platform) {
    if (player.body.touching.down)
        playerPlatform = platform;}
function collectStar(player, star) {
    soundCollectStar.play();
    star.kill();
    totalNumOfStars -= 1;}
function getFirstAid(player, aid) {
    soundGetAid.play();
    aid.kill();
    healthValue = Math.min(MAX_HEALTH, healthValue + healthAid);
    updateHealthBar();}
function playerVsEnemy(player, enemySprite, enemyObject, position) {
    if (enemySprite.body.touching.up) {
        soundHitEnemy.play();
        if (!enemyObject.flash.isRunning)
            enemyObject.flash.start();
        if (toRight)
            player.body.x += PLAYER_COLLIDE_OFFSET_X;
        else
            player.body.x -= PLAYER_COLLIDE_OFFSET_X;
        let hits = enemyObject.getHitsToBeKilled();
        enemyObject.setHitsToBeKilled(hits - 1);
        if (enemyObject.getHitsToBeKilled() <= 0) {
            enemySprite.destroy();
            enemies.splice(position, 1);}
    } else {
        soundDamaged.play();
```

```
        healthValue = Math.max(0, healthValue - damage);
        updateHealthBar();
        if (toRight)
            player.body.x -= PLAYER_COLLIDE_OFFSET_X;
        else
            player.body.x += PLAYER_COLLIDE_OFFSET_X;
        if (healthValue === 0)
            resetPlayer();
    }player.body.y -= PLAYER_COLLIDE_OFFSET_Y;}
function updateHealthBar() {
    if (healthTween)
        healthTween.stop();
    // write the required instructions to set the tween and start it
    healthTween = game.add.tween(healthBar.scale).to({x: healthValue/MAX_HEALTH}, 300, Phaser.Ea
sing.Cubic.Out);}
function resetPlayer() {
    stopPlayer();
    player.x = levelConfig.collectorStart.x;
    player.y = game.world.height - levelConfig.collectorStart.y;
    remainingTime = Math.max(0, remainingTime - playerDeathTimePenalty);
    healthValue = MAX_HEALTH;
    updateHealthBar();}
function setRemainingTime(seconds) {
    return String(Math.trunc(seconds / 60)).padStart(2, "0") + ":" +
        String(seconds % 60).padStart(2, "0");}
function updateTime() {
    remainingTime = Math.max(0, remainingTime - 1);
    hudTime.setText(setRemainingTime(remainingTime));
    if (remainingTime === 0) {
        resetInput();
        soundOutOfTime.play();
        stopPlayer();
        game.time.events.remove(timerClock);
        game.time.events.add(2500, endGame, this); }}
function stopPlayer() {
    player.animations.stop();
    player.frame = 4;}
function resetInput() {
    game.input.enabled = false;
    cursors.left.reset(true);
    cursors.right.reset(true);
    cursors.up.reset(true);
    cursors.down.reset(true);}
function endLevel() {
    if (totalNumOfStars === 0) {
        exitingLevel = true;
        resetInput();
        soundLevelPassed.play();
        stopPlayer();
```

```
                game.time.events.remove(timerClock);
                game.time.events.add(4000, nextLevel, this); }}
            function endGame() {
                clearLevel();
                goToWelcome();}
            function nextLevel() {
                clearLevel();
                levelToPlay += 1;
                if (levelToPlay > levelsData.length)
                    goToWelcome();
                else {
                    game.input.enabled = true;
                    game.state.start('play'); }}
            function clearLevel() {
                for (let i = 0, max = enemies.length; i < max; i++) {
                    enemies[i].sprite.destroy();  }
                enemies = [];
                hudGroup.removeAll(true);
                platforms.removeAll(true);
                player.destroy();
                firstAids.removeAll(true);
                stars.removeAll(true);}
            function goToWelcome() {
                game.world.setBounds(0, 0, game.width, game.height);
                game.state.start('welcome');}
```

En configure.js puedes determinar la dificultad con botones. Main.js derecha.

```
function onButtonPressed(button) {
    // Write the code for this function
    if(button === btnEasy) {
        damage = DEFAULT_DAMAGE;
        healthAid = DEFAULT_HEALTH;
        secondsToGo = DEFAULT_TIME;
        jumpsToKill = DEFAULT_JUMPS_TO_KILL;
        playerDeathTimePenalty = DEFAULT_PLAYER_DEATH_TIME_PENALTY;
    }

    else if (button === btnAvg){
        damage = DEFAULT_DAMAGE * 1.5;
        healthAid = DEFAULT_HEALTH - 2;
        secondsToGo = DEFAULT_TIME - 90;
        jumpsToKill = DEFAULT_JUMPS_TO_KILL + 1;
        playerDeathTimePenalty = DEFAULT_PLAYER_DEATH_TIME_PENALTY + 10;
    }

    else if (button === btnNgtm){
        damage = DEFAULT_DAMAGE * 2;
        healthAid = DEFAULT_HEALTH - 5;
        secondsToGo = DEFAULT_TIME - 150;
        jumpsToKill = DEFAULT_JUMPS_TO_KILL + 3;
        playerDeathTimePenalty = DEFAULT_PLAYER_DEATH_TIME_PENALTY + 25;
    }

    game.state.start('welcome');
}
```

```
let game;

let wfConfig = {
    active: function () {
        startGame();
    },

    google: {
        families: ['Rammetto One', 'Sniglet']
    },

    custom: {
        families: ['FerrumExtracondensed', 'Winkle'],
        urls: ["https://fontlibrary.org/face/ferrum", "assets/fonts/Winkle-Regular.ttf"]

    }
};

WebFont.load(wfConfig);

function startGame() {
    game = new Phaser.Game(800, 600, Phaser.CANVAS, 'platformGameStage');

    // Welcome Screen
    game.state.add('welcome', initialState);
    // About Screen
    game.state.add('about', aboutState);
    // Config Screen
    game.state.add('config', configState);
    // Play Screen
    game.state.add('play', playState);

    // Add the instruction required to start the 'welcome' state

    game.state.start('welcome');
}
```

Orden scripts en html

```
<script src='https://cdnjs.cloudflare.com
<script src="js/lib/phaser.js"></script>
<script src="js/welcome.js"></script>
<script src="js/about.js"></script>
<script src="js/configure.js"></script>
<script src="js/play.js"></script>
<script src="js/main.js"></script>
```

PR5: BASIC SHOOTER

init.js

```
const TEXT_OFFSET_HOR = 40;
const TEXT_OFFSET_VER = 40;
const SHIP_OFFSET_HOR = 150;
const SHIP_OFFSET_VER = 90;
let btnStart;
let imgUfo;
let initState = {
    preload: preloadInit,
    create: createInit
};
function preloadInit () {
    game.load.image('craft', 'assets/imgs/craft.png');
    game.load.image('ufo', 'assets/imgs/ufo.png');
}
function createInit() {
    let textI = 'Left and right cursors move the shooter,\n';
    textI += 'and also horizontal movements of the mouse.\n';
    textI += 'Spacebar and mouse clicks fire the laser cannons.';
    textI += '\n\nClick on the spacecraft to start.';
    let styleI = {font:'20px Arial', fill:'#FFFFFF'};
    let instructions = game.add.text(TEXT_OFFSET_HOR, TEXT_OFFSET_VER, textI, styleI);

    let textC = 'Credits:\n';
    textC += '* Original craft pic created by "Fran" (Desarrollo XNA).\n';
    textC += '* Original UFO pic created by "0melapics" (Freepik.com).\n';
    textC += '* Original laser pic from Phaser tutorial "Invaders".\n';
    textC += '* Blast animation from Phaser tutorial "Invaders".\n';
    textC += '* Blast sound created by "dklon" (OpenGameArt.Com).\n';
    textC += '* Laser sound created by "dklon" (OpenGameArt.Com).';
    let styleC = {font:'16px Arial', fill:'#FF0000'};
    let credits = game.add.text(TEXT_OFFSET_HOR, game.world.height-
TEXT_OFFSET_VER, textC, styleC);
    credits.anchor.setTo(0, 1);
   let posX = game.world.width-SHIP_OFFSET_HOR;
    let posY = game.world.height-SHIP_OFFSET_VER;
    posY = game.world.centerY;
    imgUfo = game.add.image(posX, posY, 'ufo');
    imgUfo.anchor.setTo(0.5, 0.5);
    imgUfo.scale.setTo(2.0);
```

```
    btnStart = game.add.button(posX, posY, 'craft', clickStart);
    btnStart.anchor.setTo(0.5, 0.5);
    btnStart.scale.setTo(2.0);
    btnStart.checkWorldBounds = true;
    btnStart.events.onOutOfBounds.add(startPlay, this);}
const FREQUENCY = 1000/30;
function clickStart() {
    btnStart.inputEnabled = false;
    game.time.events.loop(
    FREQUENCY, moveButtonAndImage, this);   }
function startPlay() {
    game.state.start('play');}
const DECREASE_Y = 8;
const DECREASE_X = 10;
function moveButtonAndImage() {
btnStart.y -= DECREASE_Y;
imgUfo.x -= DECREASE_X;}
```

play.js

```
let craft;const HUD_HEIGHT = 50;let cursors;const CRAFT_VELOCITY = 150;let stars;const LASERS_GR
OUP_SIZE = 40;let lasers;const LEFT_LASER_OFFSET_X = 11;const RIGHT_LASER_OFFSET_X = 12;const LA
SERS_OFFSET_Y = 10;const LASERS_VELOCITY = 500;let fireButton;let soundLaser;const UFOS_GROUP_SI
ZE = 200;let ufos;const TIMER_RHYTHM=0.1*Phaser.Timer.SECOND;let currentUfoProbability;let curre
ntUfoVelocity;const BLASTS_GROUP_SIZE = 30;let blasts;let soundBlast;
//HUD
let score; // Repeated declaration in hof.jslet scoreText;let level;let levelText;let lives;let
livesText;const NUM_LEVELS = 5;const LEVEL_UFO_PROBABILITY =[0.2, 0.4, 0.6, 0.8, 1.0];const LEVE
L_UFO_VELOCITY =[50, 100, 150, 200, 250];const HITS_FOR_LEVEL_CHANGE = 50;
let playState = {
    preload: preloadPlay,
    create: createPlay,
    update: updatePlay
};
function preloadPlay() {
    game.load.image('craft', 'assets/imgs/craft.png');
    game.load.image('stars','assets/imgs/stars.png');
    game.load.image('laser','assets/imgs/laser.png');
    game.load.audio('sndlaser','assets/snds/laser.wav');
    game.load.image('ufo','assets/imgs/ufo.png');
    game.load.spritesheet('blast','assets/imgs/blast.png',128,128);
    game.load.audio('sndblast','assets/snds/blast.wav');}
function createPlay() {
    score = 0;
    level = 1;
    lives = 3;
    let w = game.world.width;
```

```
    let h = game.world.height;
    stars = game.add.tileSprite(0, 0, w, h, 'stars');
    createHUD();
    createCraft();
    createKeyControls();
    createLasers(LASERS_GROUP_SIZE);
    createSounds();
    createBlasts(BLASTS_GROUP_SIZE);
    createUfos(UFOS_GROUP_SIZE);    }
function createHUD() {
    let scoreX = 5;
    let levelX = game.world.width / 2;
    let livesX = game.world.width - 5;
    let allY = game.world.height - 25;
    let styleHUD = {fontSize: '18px', fill: '#FFFFFF'};
    scoreText = game.add.text(scoreX,allY,'Score: '+score,styleHUD);
    levelText = game.add.text(levelX,allY,'Level: '+level,styleHUD);
    levelText.anchor.setTo(0.5, 0);
    livesText = game.add.text(livesX,allY,'Lives: '+lives,styleHUD);
    livesText.anchor.setTo(1, 0);}
function createBlasts(number) {
    blasts = game.add.group();
    blasts.createMultiple(number, 'blast');
    blasts.forEach(setupBlast, this);}
function setupBlast(blast) {
    blast.anchor.x = 0.5;
    blast.anchor.y = 0.5;
    blast.animations.add('blast');}
function createCraft(){
    let x = game.world.centerX;
    let y = game.world.height - HUD_HEIGHT;
    craft = game.add.sprite(x, y, 'craft');
    craft.anchor.setTo(0.5, 0.5);
    game.physics.arcade.enable(craft);
    craft.body.collideWorldBounds = true;}
function createUfos(number) {
    ufos = game.add.group();
    ufos.enableBody = true;
    ufos.createMultiple(number, 'ufo');
    ufos.callAll('events.onOutOfBounds.add',
    'events.onOutOfBounds', resetMember);
    ufos.callAll('anchor.setTo', 'anchor', 0.5, 1.0);
    ufos.setAll('checkWorldBounds', true);
    currentUfoProbability = LEVEL_UFO_PROBABILITY[level-1];
    currentUfoVelocity = LEVEL_UFO_VELOCITY[level-1];
    game.time.events.loop(TIMER_RHYTHM, activateUfo, this);}
function activateUfo() {
    if (Math.random() < currentUfoProbability) {
        let ufo = ufos.getFirstExists(false); //Si el ufo está inexistente reiniciamos sus valor
```

```
        if (ufo) {
            let gw = game.world.width;
            let uw = ufo.body.width;
            let w = gw - uw;
            let x = Math.floor(Math.random()*w);
            let z = uw / 2 + x;
            ufo.reset(z, 0);
            ufo.body.velocity.x = 0;
            ufo.body.velocity.y = currentUfoVelocity;}
    }
}

function createSounds() {
    soundLaser = game.add.audio('sndlaser');
    soundBlast = game.add.audio('sndblast');}
function createLasers(number) {
    lasers = game.add.group();
    lasers.enableBody = true;
    lasers.createMultiple(number, 'laser');
    lasers.callAll('events.onOutOfBounds.add',
    'events.onOutOfBounds', resetMember);
    lasers.callAll('anchor.setTo', 'anchor', 0.5, 1.0);
    lasers.setAll('checkWorldBounds', true);}
function resetMember(item) {
    item.kill();}
function createKeyControls(){
    cursors = game.input.keyboard.createCursorKeys();
    fireButton = game.input.keyboard.addKey(Phaser.Keyboard.SPACEBAR);}
function updatePlay() {
    game.physics.arcade.overlap(lasers,ufos,laserHitsUfo,null,this);
    game.physics.arcade.overlap(craft,ufos,ufoHitsCraft,null,this);
    stars.tilePosition.y += 1;
    manageCraftMovements();
    manageCraftShots();}
function laserHitsUfo(laser, ufo) {
    ufo.kill();
    laser.kill();
    displayBlast(ufo);
    soundBlast.play();
    score++;
    scoreText.text = 'Score: '+score;
    if (level < NUM_LEVELS &&
        score===level*HITS_FOR_LEVEL_CHANGE) {
        level++;
        levelText.text = 'Level: ' + level;
        currentUfoProbability = LEVEL_UFO_PROBABILITY[level-1];
        currentUfoVelocity = LEVEL_UFO_VELOCITY[level-1]; }
}
```

```
function ufoHitsCraft(craft, ufo) {
    ufo.kill();
    craft.kill();
    displayBlast(ufo);
    displayBlast(craft);
    soundBlast.play();
    lives--;
    livesText.text = 'Lives: '+lives;
    ufos.forEach(clearStage, this);
    lasers.forEach(clearStage, this);
    game.input.enabled = false;
    currentUfoProbability = -1;
    game.time.events.add(2000, continueGame, this);}
function clearStage(item) {
    item.kill();}
function continueGame() {
    game.input.enabled = true;
    if (lives > 0) {
        let x = game.world.centerX;
        let y = game.world.height - HUD_HEIGHT;
        craft.reset(x, y);
        cursors.left.reset();
        cursors.right.reset();
        currentUfoProbability =LEVEL_UFO_PROBABILITY[level-1];}
    else
        startHOF();}
function displayBlast(ship) {
    let blast = blasts.getFirstExists(false);
    let x = ship.body.center.x;
    let y = ship.body.center.y;
    blast.reset(x, y);
    blast.play('blast', 30, false, true);}
function manageCraftShots() {
    if (fireButton.justDown ||game.input.mousePointer.leftButton.justPressed(30))
        fireLasers();}
function fireLasers() {
    let lx = craft.x - LEFT_LASER_OFFSET_X;
    let rx = craft.x + RIGHT_LASER_OFFSET_X;
    let y = craft.y - LASERS_OFFSET_Y;
    let vy = -LASERS_VELOCITY;
    let laserLeft = shootLaser(lx, y, vy);
    let laserRight = shootLaser(rx, y, vy);
    if (laserLeft || laserRight)
        soundLaser.play();}
function shootLaser(x, y, vy) {
    let shot = lasers.getFirstExists(false);  //Si el shot está inexistente(getFirstExists(false
)) se reinicia.
    if (shot) {
        shot.reset(x, y);
```

```
        shot.body.velocity.y = vy;
    }
    return shot;
}
function manageCraftMovements() {
    craft.body.velocity.x = 0;
    if (cursors.left.isDown || game.input.speed.x < 0)
    craft.body.velocity.x = -CRAFT_VELOCITY;
    else if (cursors.right.isDown || game.input.speed.x > 0)
    craft.body.velocity.x = CRAFT_VELOCITY;}
function startHOF() {
    game.state.start('hof');}
```

hof.js

```
class HallOfFame {
    constructor(thesize = 10) {
        this.size = thesize;
        this.list = [];}
    addNewScore(newScore) {
        let i;
        for (i = 0 ; i < this.list.length ; i++)
            if (newScore > this.list[i].score)
                break;
        let instant = (new Date()).toUTCString();
        this.list.splice(i, 0, {score: newScore, date: instant});
        if (this.list.length > this.size)
            this.list.pop();
        if (i < this.size)
            return i;
        else
            return -1; }
    loadFromStorage() {
        if (localStorage.sizeHOF !== undefined)
            this.size = JSON.parse(localStorage.sizeHOF);
        if (localStorage.listHOF !== undefined)
            this.list = JSON.parse(localStorage.listHOF); }
    saveToStorage() {
        localStorage.sizeHOF = JSON.stringify(this.size);
        localStorage.listHOF = JSON.stringify(this.list); }
    resetStorage() {
        localStorage.removeItem("sizeHOF");
        localStorage.removeItem("listHOF");}
    getSize() {
        return this.size; }
    setSize(thesize) {
        if (thesize !== undefined) {
            this.size = thesize;
            while (this.list.length > this.size)
```

```
                this.list.pop();
        }}
    displayOnStage(x, y, w, h) {
        let hT = Math.floor(0.25 * h);
        let sT = Math.floor(0.25 * 0.6 * h);
        let styleT = {font:sT+'px Arial', fill:'#00FF00',
                    boundsAlignH:'center', boundsAlignV:'bottom'};
        let title = game.add.text(0, 0, 'Hall of Fame', styleT);
        title.setTextBounds(x, y, w, hT);
        let hL = Math.floor(0.75 * 0.1 * h);
        let sL = Math.floor(0.75 * 0.1 * 0.8 * h);
        let wCP = Math.floor(0.06 * w);
        let wCS = Math.floor(0.18 * w);
        let wCD = Math.floor(0.76 * w);
        let styleL = {font:sL+'px Arial', fill:'#FFFF00',
                    boundsAlignH:'right', boundsAlignV:'bottom'};
        let placeL, scoreL, dateL;
        for (let i = 0 ; i < this.list.length ; i++) {
            placeL = game.add.text(0, 0, i+1, styleL);
            placeL.setTextBounds(x, y+hT+hL*i, wCP, hL);

            scoreL = game.add.text(0, 0, this.list[i].score, styleL);
            scoreL.setTextBounds(x+wCP, y+hT+hL*i, wCS, hL);

            dateL = game.add.text(0, 0, this.list[i].date, styleL);
            dateL.setTextBounds(x+wCP+wCS, y+hT+hL*i, wCD, hL); }
    }
} // class HallOfFame
let shooterHOF;
let msgUser;
let hofState = {
    preload: preloadHOF,
    create: createHOF};
function restartPlay() {
    game.state.start('play');}
function preloadHOF() {
    shooterHOF = new HallOfFame();
    shooterHOF.loadFromStorage();
    let i = shooterHOF.addNewScore(score);
    if (i >= 0) {
        msgUser = "Congratulations! The " + (i+1) + ordinalNumAbbrev(i+1);
        msgUser += " place honours this shooting session."; }
    else
        msgUser = "Sorry! This shooting session have no place here.";
    shooterHOF.saveToStorage();
}
function ordinalNumAbbrev(n) {
    if (n > 0) {
        let last = n % 10;
```

```
        let remaining = n / 10;
        let nextToLast = remaining % 10;
        if (nextToLast === 1)
            return 'th';
        switch (last) {
            case 1: return 'st' ;;
            case 2: return 'nd' ;;
            case 3: return 'rd' ;;
            default: return 'th' ;;
        }
    }
                return ''; //MAIN.JS →
}
function createHOF() {
    game.stage.backgroundColor = "#000033";

    shooterHOF.displayOnStage(150, 0, 500, 400);

    let msg = game.add.text(0, 0, msgUser,
              {font:'27px Arial', fill:'#00FFFF',
               boundsAlignH:'center', boundsAlignV:'bottom'});
    msg.setTextBounds(0, game.world.height-150, game.world.width, 40);

    let again = game.add.text(0, 0, "Do you want to shoot again? Press 'R' to restart.",
{font:'27px Arial', fill:'#FFFFFF',boundsAlignH:'center', boundsAlignV:'bottom'});
    again.setTextBounds(0, game.world.height-80, game.world.width, 40);
    let rkey = game.input.keyboard.addKey(Phaser.Keyboard.R);
    rkey.onDown.addOnce(restartPlay, this);}
```

```
const GAME_STAGE_WIDTH = 800;
const GAME_STAGE_HEIGHT = 600;

let game = new Phaser.Game(GAME_STAGE_WIDTH, GAME_STAGE_HEIGHT, Phaser.CANVAS, 'gamestage');

// Entry point
window.onload = startGame;

function startGame() {
    game.state.add('init', initState);
    game.state.add('play', playState);
    game.state.add('hof', hofState);

    game.state.start('init');
}
```

```
<body>
    <div id="gamestage"></div>
    <script src="js/lib/phaser.js"></script>
    <script src="js/init.js"></script>
    <script src="js/play.js"></script>
    <script src="js/hof.js"></script>
    <script src="js/main.js"></script>
</body>
```

EXAMEN JUNIO 2019

EJERCICIOS HTML

```
 <link rel="stylesheet" type="text/css" href="css/estilos.css"> //...

 <div id="begin">

        <h3>PANTALLA INICIAL</h3>

        <div class="forExam">Este es un ejercicio de examen diseñado para que
demuestres tus conocimientos de HTML, CSS y (un poquito, muy poquito) de JavaScript.

        </div>

        <div class="forExam">Tienes que conseguir que tanto este párrafo como el
anterior y el siguiente se muestren de manera idéntica, tal como se indica en el
enunciado del examen. </div>

        <div class="forExam">Para conseguirlo, fíjate en los estilos que hay
definidos en el fichero <span>estilos.css</span> que encontrarás dentro de la carpeta
<span>css</span>.

        </div>

    </div>
```

```
    <div id="end">

        <h3>PANTALLA FINAL</h3>

        <div class="forExam">Tienes que aplicar el estilo anterior igualmente tanto a
este párrafocomo al siguiente dentro de lo que sería esta sección.</div>

        <div class="forExam">Este es el párrafo final. Es el último. Observa bien el
resultado al cargar la página.</div>

    </div>

    <div onclick="cambioDiv()">

        <img src="imgs/change.png" alt="cambiar"
style="width:100px;height:110px;margin-left:50%;margin-right:50%;margin-top:3em">

    </div>

    <script> function cambioDiv(){

    if(document.getElementById("begin").style.display == "none"){

        document.getElementById("begin").style.display = "inline";

        document.getElementById("end").style.display = "none";

    else{

        document.getElementById("begin").style.display = "none";

        document.getElementById("end").style.display = "inline";}}</script>
```

CSS

```
@font-face {
    font-family: 'CoelacanthRegular';
    src: url('../fonts/Coelacanth.otf');
}

body {
    background-color: slategrey;
    margin: auto;
    margin-top: 2em;
    width: 75%;
}

.forExam {
    background: #01395e;
    border-color: #b9b5b5 #b9b5b5 #272525 #272525;
    border-radius: 15px;
    box-shadow: 10px 10px 15px 5px #000000;
    border-width: 3px;
    padding: 1em 1em 1em 1em;
    font-family: 'CoelacanthRegular', normal;
    font-size: 18pt;
    text-align: center;
    color: #74c925;
    margin-bottom: 1em;
}
```

```
span {
    text-decoration: underline;
    font-weight: bold;
}

#begin{
    display: inline;
}

#end{
    display: none;
}
```

EJERCICIO 2: PHASER

1. PLASMAS

En la function create:

```
createPlasmas(PLASMA_GROUP_SIZE);//CAMBIO
function createPlasmas(number) {//CAMBIO
    plasmas = game.add.group();
    plasmas.enableBody = true;
```

```
    plasmas.createMultiple(number, 'plasma');
    plasmas.callAll('events.onOutOfBounds.add', 'events.onOutOfBounds', resetMember);
    plasmas.callAll('anchor.setTo', 'anchor', 0.5, 1.0);
    plasmas.setAll('checkWorldBounds', true);
    game.time.events.loop(TIMER_RHYTHM, activatePlasma, this);
}
function activatePlasma() {//CAMBIO
    if (Math.random() < 0.1) {
        let plasma = plasmas.getFirstExists(false);
        if (plasma) {
            //posiciones
            let gw = game.world.width;
            let gh = game.world.height;
            let x = Math.floor(Math.random() * (gw+gh/2)-gh/2);
            let y = 0;
            plasma.reset(x,y);
            game.physics.arcade.enable(plasma);
            //velocidades
            let v= VEL_PLASMA;
            let d = Math.sqrt(Math.pow(x-craft.x,2)+Math.pow(y-craft.y,2));
            let vx = -(x-craft.x)/d*v;
            let vy = -(y-craft.y)/d*v;
            plasma.body.velocity.x = vx;
            plasma.body.velocity.y = vy;
        }
    }
}
function updatePlay() {
    game.physics.arcade.overlap(lasers, ufos, laserHitsUfo, null, this);
    game.physics.arcade.overlap(craft, ufos, ufoHitsCraft, null, this);
    game.physics.arcade.overlap(craft, plasmas, plasmaHitsCraft, null, this); //CAMBI
    stars.tilePosition.y += 1;
    craft.body.velocity.x = 0;
    manageCraftMovements();
    if(!inhibition){ //CAMBIO
        manageCraftShots();
    }
}
function plasmaHitsCraft(craft, plasma) { //CAMBIO
    plasma.kill();
    inhibition = true;
    game.time.events.add(2000, inhibitionchange, this);
}
```

2- END

```
function startEnd() {
    game.state.start('end'); //CAMBIO
}

const POS_TEXT_Y= 250;
```

```
let endState = { //Esto estaba desde 0, así que todo lo escrito es nuevo
    preload: preloadEnd,
    create: createEnd,
    update: updateEnd};
function preloadEnd(){
    game.load.image('craft', 'assets/imgs/craft.png');}
function createEnd(){
    let textText1 = "Last score:";
    let textText2 = score;
    let styleText1 = {font: '35px Arial', fill: 'yellow'};
    let styleText2 = {font: '75px Arial', fill: 'orange'};
    game.add.text(POS_TEXT_X, POS_TEXT_Y, textText1, styleText1);
    game.add.text(POS_TEXT_X+20, POS_TEXT_Y+50, textText2, styleText2);
    craft = game.add.sprite(0,0, 'craft');
    craft.anchor.setTo(0.5, 0.5);
    craft.angle= 90;
    craft.scale.setTo(2,2);
    craft.x = craft.width;
    craft.y = craft.height}
function updateEnd(){
    if(craft.angle == 90){
        if(craft.width+craft.x >= game.world.width){
            craft.angle = -180;}
        else{
            craft.x += 5;}}
    else if(craft.angle == -180){
        if(craft.height+craft.y >= game.world.height){
            craft.angle = 270;}
        else{
            craft.y += 5;}}
    else if(craft.angle == -90){
        if(craft.x-craft.width <= 0){
            craft.angle = 0; }
        else{
            craft.x -= 5;}}
    else if(craft.angle == 0){
        if(craft.y-craft.height <= 0){
            craft.angle = 90;}
        else{
            craft.y -= 5; }  }      }
```

EJERCICIOS JAVASCRIPT

```
class Ship {
    constructor(x, y, width, height, color, img = null) { //CAMBIO
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.color = color;
```

```
        this.speedX = 0;
        this.speedY = 0;
        this.img = img; //CAMBIO}
    setSpeedX(speedX) {
        this.speedX = speedX;}
    setSpeedY(speedY) {
        this.speedY = speedY;}
    render(ctx) { //CAMBIO
        if (this.img == null){
            ctx.fillStyle = this.color;
            ctx.fillRect(this.x, this.y, this.width, this.height);}
        else{
            ctx.drawImage(this.img, this.x, this.y, this.width, this.height);} }
    move() { //CAMBIO
        this.x += this.speedX;
        this.y += this.speedY;
        if(this.y <=0){
            this.speedY = -this.speedY;}
        if(this.y >=canvas.height-this.height){
            this.speedY = -this.speedY;}}
   setIntoArea(endX, endY) {
        this.x = Math.min(Math.max(0, this.x), (endX - this.width));
        this.y = Math.min(Math.max(0, this.y), (endY - this.height));}
    crashWith(obj) {
        // detect collision with the bounding box algorithm
        let myleft = this.x;
        let myright = this.x + this.width;
        let mytop = this.y;
        let mybottom = this.y + this.height;
        let otherleft = obj.x;
        let otherright = obj.x + obj.width;
        let othertop = obj.y;
        let otherbottom = obj.y + obj.height;
        let crash = true;
        if ((mybottom < othertop) || (mytop > otherbottom) || (myright < otherleft) ||
            (myleft > otherright)) {
            crash = false; }
        return crash; }
    setX(newX){ //CAMBIO
        this.x = newX;}
   setY(newY){ //CAMBIO
        this.y = newY;}}
class GameCanvas {
    constructor(_canvas, _ship, _ufos, _livesleft) {
        this.canvas = _canvas;
        this.ship = _ship;
        this.ufos = _ufos;
        this.context = null;
        this.interval = null;
```

```
        this.livesleft = _livesleft; }
    initialise(canvasWidth, canvasHeight) {
        this.canvas.width = canvasWidth;
        this.canvas.height = canvasHeight;
        this.context = this.canvas.getContext("2d");
        this.interval = setInterval(updateGame, 1000 / FPS); }
    render() {
        for (const ufo of this.ufos) {
            ufo.render(this.context);}
        this.ship.render(this.context);}
   renderLives(){//CAMBIO (ESTO NO SÉ CÓMO HACERLO Y PENSADO EMPEZANDO DE ESTA MANERA IDK)
        for(let i = 0; i < lives; i++){
            let imageLives = document.createElement('img');
            imageLives.src = 'imgs/spaceship.png';
            imageLives.style.width = WIDTH_LIVE;
            imageLives.style.height = HEIGHT_LIVE;
            imageLives.style.left = i + 10 + 'px'
            this.livesleft.appendChild(imageLives);} }
    removeLive(){
        this.livesleft.removeChild(livesleft.lastChild);}
    clear() {
        this.context.clearRect(0, 0, this.canvas.width, this.canvas.height); }
    addUfo(ufo) {
        this.ufos.push(ufo);}
    removeUfo(i) {
        this.ufos.splice(i, 1); }
    removeAllUfos() {
        for(let i = 0; i < this.ufos.length; i++){
            this.removeUfo(i); }
    }
}
let canvas = document.getElementById("canvasGame"); //CAMBIO
let canvasWidth = canvas.width; //CAMBIO
let canvasHeight = canvas.height; //CAMBIO
let livesleft = document.getElementById("livesleft"); //CAMBIO
let image = new Image(); //CAMBIO
image.src = 'imgs/spaceship.png'; //CAMBIO
let theShip = new Ship(0, canvasHeight / 2, SHIP_WIDTH, SHIP_HEIGHT, DEFAULT_COLOR_SH
IP, image); //CAMBIO
let rightArrowPressed = false,
    leftArrowPressed = false,
    upArrowPressed = false,
    downArrowPressed = false;
let seconds, timeout, theChrono;
let continueGame = true;
let gameArea = new GameCanvas(canvas, theShip, [], livesleft); //CAMBIO
let lives = LIVES_BEGIN; //CAMBIO
```

EN UPDATE:

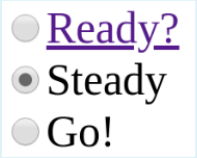
```
if (seconds % SECONDS_UFO2 === 0) { //CAMBIO
            let chance = Math.random();
            // if an ufo has to be generated
            if (chance < PROBABILITY_UFO2) {
                // random position in Y axis
let posY = Math.floor(Math.random() * ((canvasHeight - UFO_HEIGHT)*4/5) + ((canvasHei
ght - UFO_HEIGHT)/5));
                let imageUfo = new Image();
                imageUfo.src = 'imgs/nave_alien.png';
let ufo = new Ship(canvasWidth, posY, UFO_WIDTH, UFO_HEIGHT, DEFAULT_COLOR_UFOS, imageUfo);
                // random speed
                let vX = Math.random() *15 +6;
                let vY = Math.random() *15 +6;
                if(Math.random() <= 0.5) vY = -vY;
                ufo.setSpeedX(-vX);
                ufo.setSpeedY(vY);
                gameArea.addUfo(ufo);
            }
        }
        let minutes = Math.floor(seconds / 60);
        let secondsToShow = seconds % 60;
theChrono.innerHTML = CHRONO_MSG + " " + String(minutes).padStart(2, "0") + ":" +
            String(secondsToShow).padStart(2, "0");
        timeout = window.setTimeout(updateChrono, 1000);
    }
```

CSS:

```
#livesleft{ /*CAMBIO*/
    margin-left: 2em;
    margin-top: 1em;
    padding: 0.2em;
    vertical-align: middle;
    display: inline-block;
}
```

EXAMEN 2020

Escribe el código en HTML para obtener tener estos tres botones de radio:



Para simplificar, o por si alguna parte no sabes/puedes hacerla, puedes proceder en este orden:

- Que aparezcan las tres opciones, sin preocuparte de otras consideraciones.
- Que la primera de ellas sea un enlace a la página *http://www.uji.es*.
- Que las opciones sean mútuamente excluyentes.

Pista: considera el atributo *name*.

```
<h1>Ejercicio 1</h1>
```

```
<form action="">
  <p>Botones:</p>
  <input type="radio" id="ready" name="position" value="ready"> <a
href="http://www.uji.es">Ready?</a><br>
  <input type="radio" id="steady" name="position" value="steady">
Steady? <br>
  <input type="radio" id="go" name="position" value="go"> Go!
</form>

</body>
</html>


a:link {
    color: #006666;
}

a:visited {
    color: #4df89c;
}

a:hover {
    color: #290333;
}
```

Tenemos una página web que contiene un elemento `<div>` vacío, identificado con el **id** "fillme":

```
<div id="fillme">
</div>
```

Escribe las instrucciones JavaScript requeridas para manipular el DOM de esta página web y hacer que los contenidos de este elemento `<div>` sean los siguientes:

```
<div id="fillme">
    <p style="font-family: Arial; font-size: 16pt; color: blue;">
              A continuación, una lista numerada</p>
    <ol>
       <li>Primer ítem</li>
       <li>Segundo ítem</li>
    </ol>
</div>
```

```
function init() {
    let scene = document.getElementById("fillme");
    let paragraph = document.createElement("p");
    scene.appendChild(paragraph);
    let letters = document.createTextNode("A continuación, una lista
numerada");
    paragraph.appendChild(letters);
    let style = document.createAttribute("style");
    style.value = "font-family: Arial; font-size: 16pt; color: blue;";
    paragraph.setAttributeNode(style);
    let lists = document.createElement("ol");
    paragraph.appendChild(lists);
    let list1 = document.createElement("li");
    lists.appendChild(list1);
    let list1text = document.createTextNode("Primer ítem");
    list1.appendChild(list1text);
    let list2 = document.createElement("li");
    lists.appendChild(list2);
    let list2text = document.createTextNode("Segundo ítem");
    list2.appendChild(list2text);
};
```

En un juego web se dispone de una clase para almacenar los elementos de un HUD (Head-Up Display) y mostrarlos en una determinada posición. Al constructor de dicha clase se le pasan como parámetros las coordenadas X e Y a partir de las cuales se desea dibujar dicho HUD. Los elementos del HUD son dos: imágenes representando el número de vidas que le quedan al jugador y un texto que indica la puntuación del juego (fíjate en la imagen que acompaña a este ejercicio). Éste es el constructor de la clase:

```
class Hud {
    constructor(_posX, _posY) {
        this.x = _posX;
        this.y = _posY;
        // Load image for HUD
        this.hudShip = new Image();
        this.hudShip.src = "../imgs/spaceshipHUD.png";
    }
}
```

Se desea implementar un método, `render`, que, mediante instrucciones puramente JavaScript, permita dibujar el HUD tal como se muestra en la imagen. Este método recibirá tres parámetros: **el contexto del canvas** donde se dibuja, **el número de vidas** que le quedan al jugador y **la puntuación** que lleva en el juego:

```
render(ctx, lives, score) {
    let positionX = this.x;
    // FOR LOOP TO DRAW THE LIVES
    // TWO INSTRUCTIONS TO DRAW THE SCORE (TEXT)
}
```

Escribe el bucle `for` que dibuja las vidas (**usarás el contexto del canvas y la imagen almacenada en this.hudShip**) teniendo en cuenta que se han definido dos constantes, `HUD_SHIP_WIDTH` y `HUD_SHIP_HEIGHT`, que determinan, respectivamente, el **ancho** y el **alto** con el que se dibujará **la imagen de cada vida**. A continuación, escribe las dos instrucciones JavaScript para dibujar el texto con la puntuación (véase la imagen adjunta) teniendo en cuenta que se ha usado una fuente **bold, de 14 pt. y de tipo Arial**. Asume que la variable `score` almacena un número entero. La separación entre cada imagen de las vidas (y de la última vida con el texto) es de exactamente `HUD_SHIP_WIDTH / 2`.

```
render() {
        for (const obstacle of this.obstacles) {
            obstacle.render(this.context);
            this.rival.render(this.context);
        }
        this.hero.render(this.context);
        for (const life of this.livesHUD) {
            life.render(this.context);
        }
    }

class Hud {
    constructor(_posX, _posY) {
        this.x = _posX;
        this.y = _posY;
        // Load image for HUD
        this.hudShip = new Image();
        this.hudShip.src = "images/nave.png";
    }

    render(ctx, lives, score) {
        let positionX = this.x; // "DRAWING X"

        for (let i = 0; i < lives; i++) { // FOR LOOP TO DRAW THE LIVES
            ctx.drawImage(this.hudShip, positionX, this.y);
            positionX += HUD_SHIP_WIDTH + HUD_SHIP_WIDTH/2;
        }

        ctx.font = 'bold 14pt Arial'; // TWO INSTRUCTIONS TO DRAW THE SCORE
(TEXT)
        ctx.fillText('Score: ' + score, positionX, this.Y);
    }
}

let livesArray = [];

for (let i = 0; i < LIVESNUMBER; i++) {
    livesArray.push(new SquaredForm(GAME_AREA_WIDTH / 2 -25 + (25 * i), 10,
(SQUARE_SIZE/4), (SQUARE_SIZE/4), "#FF00D4"));
}

function startGame() {
```

```
    let radioButtons = document.getElementsByName("level");
    let choice, msg;
    for (const button of radioButtons) {
        if (button.checked) {
            choice = parseInt(button.value);
            break;
        }
    }

    FPS = FPS * choice;

    gameArea.initialise();
    gameArea.render();

    window.document.addEventListener("keydown", handlerOne);
    window.document.addEventListener("keyup", handlerTwo);

    seconds = 0;
    vidasActualizables = LIVESNUMBER;
    timeout = window.setTimeout(updateChrono, 1000);
    theChrono = document.getElementById("chrono");
}
```

En la función de update...

```
if (collision) {
        console.log("llega");
        if (seconds > 15) seconds -= 15;
        else seconds = 0;
        // Life count
        let vidasPadre = document.getElementById("vidasPadre");
        vidasPadre.childNodes[vidasActualizables].removeChild;
        livesArray[vidasActualizables-1].splice; // Debería de borrarse el
cuadrado
        vidasActualizables--;
        if (vidasActualizables == 0){
            endGame();
        }
        // Position restoration
        theSquare.x = 0, theSquare.y = GAME_AREA_HEIGHT / 2;
    }
    // Else, the game continues
```

En un juego web desarrollado con **Phaser** se pretende realizar el diseño de los niveles empleando código JSON. En cada uno de los niveles habrá una serie de enemigos, *"hechiceros"*, cada uno de los cuales aparecerá en una determinada coordenada, *"posX"* (es un número entero), y tendrá una serie de *"hechizos"* y *"conjuros"*.

Cada hechizo tiene dos propiedades: *"tiempo"* (es un número entero), que indica la duración de cada hechizo, y *"damage"* (es un número flotante), que indica el daño infligido por el hechizo al jugador. Los conjuros solo tienen una propiedad, *"power"* (es un número entero), que indica el nivel de poder que tiene un conjuro determinado.

Escribe el **código JSON** para indicar que en un nivel determinado hay 2 hechiceros:

- El primer hechicero está en la *"posX"* 400 y tiene un hechizo, con *"tiempo"* 20 y *"damage"* 3.5, y dos conjuros, con valores *"power"* de 4 y 6.
- El segundo hechicero está en la *"posX"* 850 y tiene dos hechizos, con valores *"tiempo"* de 30 y 50, y valores *"damage"* 4.5 y 6.0, y un conjuro, con valor de *"power"* igual a 10.

```
  "collectorStart": {"x": 32, "y": 90},
  "ground": {"enemies": [ {"x": 240, "right": 0},
                          {"x": 880, "right": 0}
                ],
                "aids": [ {"x": 470}, {"x": 955}
                ],
                "stars": [ {"x": 100}, {"x": 190}, {"x": 840}
                ]
    },
    "platformData": [
      {"x": 0, "y": 350,
       "enemies": [ {"x": 200, "right": 1}
       ],
       "aids": [ {"x": 20}
       ],
       "stars": [ {"x": 100}
       ]
      },
      {"x": 420, "y": 500,
       "enemies": [
       ],
       "aids": [ {"x": 600}
       ],
       "stars": [ {"x": 770}
       ]
      `
```

```
,,
{"x": 500, "y": 200,
 "enemies": [ {"x": 550, "right": 1}
 ],
 "aids": [ {"x": 800}
 ],
 "stars": [ {"x": 650}, {"x": 730}
 ]
},
{"x": 800, "y": 650,
 "enemies": [ {"x": 900, "right": 0}
 ],
 "aids": [
 ],
 "stars": [ {"x": 1050}
 ]
}
]
```

La función **createCraft()** de la práctica 5 incorpora a la escena la nave espacial al inicio del juego. Para ello, en un momento dado y tras cargar el correspondiente fichero de imagen asociado a la etiqueta **'craft'** y configurar los valores adecuados de **x** e **y**, ejecuta la instrucción (véase *Your turn 3*):

```
craft = game.add.sprite(x, y, 'craft');
```

¿Sería posible sustituir esta instrucción por la siguiente:

```
craft = game.add.image(x, y, 'craft');
```

y, al mismo tiempo, poder conservar toda la funcionalidad y el resto del código actual relativo a la nave espacial en el juego?

Tal y como el código está estructurado (además de por mera funcionalidad), sería incorrecto tratar de sustituir la instrucción "craft = game.add.sprite(x, y, 'craft');" por "craft = game.add.image(x, y, 'craft');". Esto se debe a que los sprites y las imágenes, si bien visualmente comparten ciertas similitudes, distan mucho de funcionar de la misma manera.

Una imagen es un objeto (relativamente ligero) que podemos utilizar para mostrar algo por pantalla, siempre que ese "algo" no precise de animaciones o físicas. Las imágenes se pueden rotar, escalar, recortar, así como recibir eventos, lo que las hace una muy buena solución para mostrar logos, fondos o botones.

Un sprite, por contrapartida, forma parte del núcleo jugable, pudiéndose asimilar a los gameObjects de otros programas y motores como Unity. Tienen un uso mucho más extendido, y esto se debe a que contienen propiedades adicionales que dan coherencia al gameplay, ya sean físicas (sprite.body), de entrada (sprite.input), de eventos (sprite.events) o de animaciones (sprite.animations), entre muchas otras. Su forma más básica consiste en un set de coordenadas y una textura, que se renderiza sobre un canvas.

Cambiar el sprite de "craft" por una imagen, por ende, significaría echar por tierra todo el trabajo de animación y físicas empleado para la construcción de "craft", por lo que su sustitución o intercambio conllevaría un proceso algo más largo que una mera modificación en la instrucción.

PR2

```
class Weapon {
    constructor(c2d,filename,l,t) {
        this.canvas2d = c2d;
        this.image = new Image();
        this.image.myparent = this;
        this.image.src = filename;
        this.image.onload = function () {
        let m = this.myparent;
        m.canvas2d.drawImage(this,l,t);
        m.width = this.naturalWidth;
        };
        this.left = l;
        this.top = t;
        this.width;
        };
        clear() {
        let l = this.left, t = this.top;
        let w = this.image.naturalWidth;
        let h = this.image.naturalHeight;
        this.canvas2d.clearRect(l,t,w,h);
        };
        move(n) {
        this.left += n;
        let i = this.image;
        let l = this.left;
        let t = this.top;
        this.canvas2d.drawImage(i,l,t);
        };
```

```
};

let colors = ["cyan","green","gray","blue"];
let cIndex = 0;

let clash = {shield:undefined, mjolnir:undefined};
clash.timer = undefined;

clash.changeBackgroungColor = function () {
    const LEAP = 7;
    const LONG_PERIOD = 20;
    const SHORT_PERIOD = 1;
    let period;
    let sh = clash.shield, mj = clash.mjolnir;
    if (sh.left + 2 * sh.width > mj.left)
        period = SHORT_PERIOD;
    else
        period = LONG_PERIOD;
    if (sh.left % period === period - 1) {
        cIndex = (cIndex + LEAP) % colors.length;
        let bg = document.getElementById("battlefield");
        bg.style.backgroundColor = colors[cIndex];
    }
};

clash.advanceWeapons = function () {
    const SHIELD_MOVE = 1;
    const MJOLNIR_MOVE = -2;
    clash.shield.clear();
    clash.mjolnir.clear();
    clash.shield.move(SHIELD_MOVE);
    clash.mjolnir.move(MJOLNIR_MOVE);
    let sh = clash.shield, mj = clash.mjolnir;
    if (sh.left + sh.width / 2 > mj.left)
    clearInterval(clash.timer);
};

/*
clash.advanceWeapons = function () {
    const SHIELD_MOVE = 1;
    const MJOLNIR_MOVE = -2;
    clash.shield.move(SHIELD_MOVE);
    clash.mjolnir.move(MJOLNIR_MOVE);
    let sh = clash.shield, mj = clash.mjolnir;
    if (sh.left + sh.width / 2 > mj.left)
        clearInterval(clash.timer);
};
*/
clash.advancing = false;

clash.animate = function () {
    if (clash.advancing)
        clash.advanceWeapons();
    clash.changeBackgroungColor();
};

clash.initWeapons = function () {
```

```
    clash.cv = document.getElementById("scene");
    let ctx = clash.cv.getContext("2d");
    let sh = "images/CaptainAmericaShield.png";
    clash.shield = new Weapon(ctx,sh,0,260);
    let mj = "images/ThorMjolnir.png";
    clash.mjolnir = new Weapon(ctx,mj,651,250);
};

/*
clash.initWeapons = function () {
    let sh = document.getElementById("capshield");
    clash.shield = new Weapon(sh);
    let mj = document.getElementById("mjolnir");
    clash.mjolnir = new Weapon(mj);
};
*/

clash.randomArrayIndex=function(color){
    let rand= random(0,color.length)
    let bg=document.getElementById("battlefield");
    bg.style.backgroundColor = colors[rand];
    initBackground();
};

function initBackground() {
    for (let i = 0 ; i < colors.length ; i += 3) {
        colors.splice(i,0,"light"+colors[i]);
        colors.splice(i+2,0,"dark"+colors[i+1]);
    } // colors ===
["lightcyan","cyan","darkcyan","lightgreen","green","darkgreen"...
   //let decIndex = Math.random() * colors.length;
   // cIndex = Math.floor(decIndex);

};

function entryPoint() {
    initBackground();
    clash.initWeapons();
    clash.mouseOverOutCanvas();
    const T = 1000 / 50; // 20 ms
    clash.timer = setInterval(clash.animate,T);
};

clash.mouseOverOutCanvas = function () {
    clash.cv.onmouseover = function () {
    clash.advancing = true;
    };
    clash.cv.addEventListener("mouseout",
    function() {
    clash.advancing = false;
    });
};


// Entry point
window.onload = entryPoint;
```