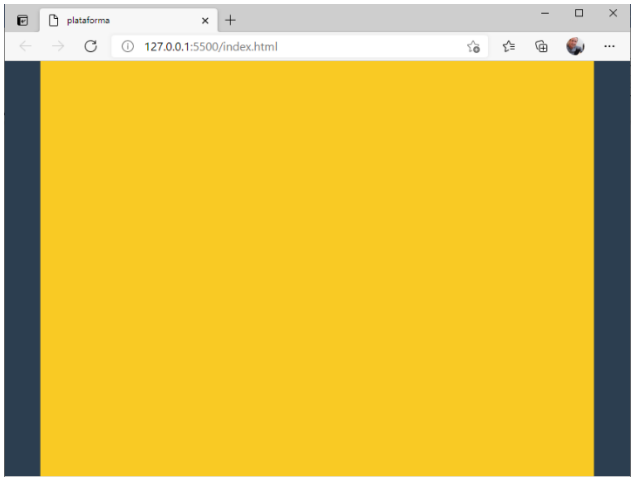
Elaboración de un juego sencillo con Phaser3 (html5, CSS y JavaScript)

El profesor te facilitará un archivo comprimido con un proyecto de “Phaser3” (tal y como los que has estado haciendo en clase). Descomprime el archivo y abre la carpeta con **Visual Studio Code**.

Dispones de la escena “Preload.js” en la que se carga el fondo (que es el cielo del juego):



Vas a precargar las siguientes imágenes:

```
this.load.image('sky', 'sky.png');
        //Cargar imágenes: plataforma, estrella y bomba
      this.load.image('ground', 'platform.png');
      this.load.image('wall', 'wall.png');
      this.load.image('star', 'star.png');
      this.load.image('bomb', 'bomb.png');
      this.load.image('life', 'life.png');
```

También vas a precargar la hoja de sprites “**dude.png”** cuyos datos son:

frameWidth : 32, frameHeight: 48

```
//Cargar spritesheet del personaje
        this.load.spritesheet('dude',
            'dude.png',
            { frameWidth: 32, frameHeight: 48 }
        );
```

En la carpeta “scenes” crea un archivo “Play.js” como una clase que extiende de Phaser:

```
class Play extends Phaser.Scene {
    constructor() {
        super('Play');
    }
```

```
    preload() {
        console.log('Play');

    }

    create() {

    }

    update(){

    }
}
export default Play;
```

Cuando se complete la precarga de la escena “Bootloader.js” haz que se ejecute la escena “Play.js”. Para eso debes importarla con :

Import Play from ‘./scenes/Play.js’;

En el archivo de configuración **“Main.js”** se debe añadir la escena “**Play**”:

```
 scene: [
        Bootloader,
        Play
    ]
```

Y en el evento ‘**complete**’ del método **preload** escribir:

this.scene.start(‘Play’);

Ahora, en el método “créate” de la escena “Play.js” debes añadir las imágenes del cielo y de la estrella:

```
this.add.image(this.scale.width / 2, this.scale.height / 2, 'sky');
this.add.image(300, 400, 'star');
```

Añadimos los sonidos:

```
//Añadir sonidos
        this.load.audio('pop', 'pop.mp3');
        this.load.audio('draw', 'draw.mp3');
        this.load.audio('sonidoFondo', 'bongojam_f.mp3');
```

A continuación cargamos la fuente (usaremos una fuente con un atlas de fuentes; archivo “json”):

```
this.load.image('font', 'font/font.png');
this.load.json('fontData', 'font/font.json');
```

Para poder utilizar la fuente disponible y que se facilita en la carpeta del proyecto debemos incluirla en el método “preload” cuando finalice de cargar, usando para ello el evento “complete” y ejecutando la siguiente función anónima:

```
 //Cuando se complete la precarga
this.load.on('complete', () => {
        console.log('Load complete');

        const fontData = this.cache.json.get('fontData');
        this.cache.bitmapFont.add('pixelFont', Phaser.GameObjects.RetroFont.Parse
        (this, fontData));

        this.scene.start('Play');
});
```

Una vez cargadas las imágenes, vamos a crear las plataformas por las que se moverá el personaje. En la función o método “créate” de la escena “Play” cargamos estas plataformas como un grupo estático con físicas de Arcade:

```
//Grupo de plataformas
        this.platforms = this.physics.add.staticGroup();
        this.platforms.create(400, 568, 'ground').setScale(2).refreshBody
();
        this.platforms.create(600, 400, 'ground');
        this.platforms.create(50, 250, 'ground');
        this.platforms.create(750, 220, 'ground');
        this.platforms.create(this.scale.width -15, 0, 'wall')
            .setOrigin(0)
            .setScale(.5, 5)
            .refreshBody();
        this.platforms.create(0, 0, 'wall')
        .setOrigin(0)
        .setScale(.5, 5)
        .refreshBody();
```

Vamos a añadir al personaje. Para eso, creamos en la carpeta “scenes” el archivo “Player.js” que va a ser una clase que extiende de la clase “Sprite” de “Phaser” y cuyo contenido será:

```
class Player extends Phaser.GameObjects.Sprite {
    constructor(config) {
        super(config.scene, config.x, config.y, 'dude');

        this.scene = config.scene;
        this.scene.add.existing(this);
        this.scene.physics.world.enable(this);
    }
}
export default Player;
```

En la escena “Play.js” cargaremos el persona y crearemos las colisiones de este con el mundo:

```
//Añadir al personaje
```

```
        this.personaje = new Player({
            scene: this,
            x: 100,
            y: 450
        });
//Creamos colisiones con el personaje de las plataformas
this.physics.add.collider([this.personaje], this.platforms);
```

En el “constructor” del archivo “Player.js” tras las configuraciones hechas anteriormente creamos las animaciones:

```
        //Crear animaciones
        this.anims.create({
            key: 'left',
            frames: this.anims.generateFrameNumbers('dude', {
                start: 0,
                end: 3
            }),
            frameRate: 10,
            repeat: -1
        });
        this.anims.create({
            key: 'turn',
            frames: [ { key: 'dude', frame: 4 } ],
            frameRate: 20
        });
        this.anims.create({
            key: 'right',
            frames: this.anims.generateFrameNumbers('dude', { start: 5, e
nd: 8 }),
            frameRate: 10,
            repeat: -1
        });
```

Añadimos también un pequeño efecto de rebote al caer:

```
this.body.setBounce(0.2);
```

Añadimos las teclas cursor:

```
//Creamos las teclas cursor
this.cursors = this.scene.input.keyboard.createCursorKeys();
```

y en la función “update” del Player creamos las condiciones para caminar hacia la derecha, hacia la izquierda, cuando está parado o saltando:

```
if (this.cursors.left.isDown){
            this.body.setVelocityX(-160);
            this.anims.play('left', true);
        }else if (this.cursors.right.isDown){
```

```
            this.body.setVelocityX(160);
            this.anims.play('right', true);
        }else{
            this.body.setVelocityX(0);
            this.anims.play('turn');
        }if (this.cursors.up.isDown && this.body.touching.down){
            this.body.setVelocityY(-330);
        }
```

Pero esta función “update” no se va a ejecutar si no la actualizamos en la función “update” de la escena “Play.js”:

```
update(){
        this.personaje.update();
        }
    }
```

Si pruebas el salto y no alcanza las plataformas o las estrellas, prueba a poner un valor mayor (en negativo):

```
if (this.cursors.up.isDown && this.body.touching.down){
            this.body.setVelocityY(-430);
        }
```

Es el momento de darle a nuestro pequeño juego un propósito. Dejemos caer una pizca de estrellas en la escena y permitamos que el personaje las recoja. Para lograr esto crearemos un nuevo archivo llamado “Starts.js” dentro de la carpeta “scenes” con el siguiente contenido:

```
class Stars extends Phaser.Physics.Arcade.Group{
    constructor(config){
        super(config.physicsworld, config.scene);
    }
}
export default Stars;
```

Añadimos una función para agregar 11 estrellas a la escena para que el personaje las recoja:

```
class Stars extends Phaser.Physics.Arcade.Group{
    constructor(config){
        super(config.physicsworld, config.scene);
        this.addStar();
        this.children.iterate(child => {
            child.setBounceY(Phaser.Math.FloatBetween(0.4, 0.8));
        });
        //Comprobar colisión con estrellas
        this.hitDelay = false;
    }
    addStar(){
        this.create(
            Phaser.Math.Between(10, this.scene.scale.width -10),
            Phaser.Math.Between(10, this.scene.scale.height -70),
            'star'
        );
```

```
    }
}
export default Stars;
```

con this.children.iterate(… se recorre todos los "hijos" del grupo y se le asigna a través de la función y el parámetro "child", la propiedad "setBounceY" (configurar el rebote en Y) en un valor aleatorio entre dos: 0.4 y 0.8.

Finalmente, agrego el grupo de estrellas a la escena "Play.js" antes o después de haber agregado al personaje:

```
//Añadimos las estrellas
        this.star = new Stars({
            physicsworld: this.physics.world,
            scene: this
        });
```

Comento la línea en la que agregué la imagen de la estrella porque no nos sirve ahora:

```
//this.add.image(300, 400, 'star');
```

Para que las estrellas no se caigan y se queden en las plataformas, hay que incluirlas en la colisión con estas (en el arreglo o array):

```
this.physics.add.collider([this.personaje, this.star], this.platforms);
```

Superposición entre estrella y personaje:

En la escena "Play.js" añadimos la función "overlap" que a su vez ejecuta una función anónima (array Function) que llama a la función "destroyItem" que se configura en el archivo "Star.js" de la estrella.

```
//Colisión entre el personaje y las estrellas
        this.physics.add.overlap(this.personaje, this.star, () =>{
            this.sound.play('pop');
            this.registry.events.emit('update_points');
            this.star.destroyItem();
        });
```

```
destroyItem(){
        this.children.entries[0].destroy();
        this.addStar();
    }
```

NOTA: Esta función se crea en el archivo de la estrella (Star.js)

Enemigos u obstáculos:

Creamos el archivo "Bombs.js" dentro de la carpeta "scenes" cuyo contenido inicial es:

```
class Bombs extends Phaser.Physics.Arcade.Group{
```

```
    constructor(config){
        super(config.physicsworld, config.scene);
    }
}
export default Bombs;
```

En la escena “Play.js” agregamos las bombas:

```
//Añadimos las bombas
this.bombsGroup = new Bombs({
physicsworld: this.physics.world,
scene: this
});
```

Pero la bomba se cae porque hay que incluirla en las colisiones con el mundo (En el archivo “Play.js”)

```
//Creamos colisiones con el personaje de las plataformas
this.physics.add.collider([this.personaje, this.star, this.bombsGroup], t
his.platforms);
```

Volvemos al archivo “Bombs.js” y añado el código para que rebote **después del constructor**. Vamos a agregar una colisión redondeada y que se desplace en función a hacia donde vaya girando. Para lo cual vamos a modificar varias propiedades del grupo.

```
addBomb(){
        this.create(100, 100, 'bomb').setDepth(2)
        .setBounce(1)
        .setCircle(5)
        .setVelocityX(
            Phaser.Math.Between(0, 1) ? 100 : -100
        )
        .setGravityY(-180);
    }
```

Añado en este archivo el método “update” para que las bombas reboten y giren:

```
update(){
        this.children.iterate( bomb => {
            if(bomb.body.velocity.x < 0){
                bomb.setAngularVelocity(-300);
            } else{
                bomb.setAngularVelocity(300);
            }
        });
    }
```

Para que el método “update” de las bombas se actualice en la escena “Play.js” debemos incluirlo en el método “update” de esta última escena (**escena “play.js”)**:

```
update(){
        this.personaje.update();
        this.bombsGroup.update();
        }
```

Vamos a programar la superposición (“overlap”) entre el personaje y las bombas para que cuando choque se ejecute la función o método “bombCollision()” que se programará en el archivo “**Player.js**”.

```
//Colisión con las bombas
       this.physics.add.overlap(this.personaje, this.bombsGroup, () =>{
           this.personaje.bombCollision();
       });
```

En el archivo “**Player.js**” creamos las variables (“life” para el número de vidas):

```
//Variable para que sólo se ejecute una vez la superposición
//del personaje con la bomba
this.hitDelay = false;
//Variable para las vidas
this.life = 3;
```

y el método (tras la función “update()”):

```
bombCollision(){
       if(!this.hitDelay){
           this.hitDelay = true;

           this.scene.sound.play('draw');
           this.life--;
           this.scene.registry.events.emit('remove_life');

           if(this.life === 0){
               this.scene.registry.events.emit('game_over');
           }
           this.setTint(0x1abc9c);
           this.scene.time.addEvent({
               delay: 300,
               callback: () =>{
                   this.hitDelay = false;
                   this.clearTint();
               }
           });
       }
   }
```

Reproducimos el sonido “draw”, se disminuye el número de vidas en una unidad y se emite un evento que vamos a registrar con el nombre de ‘remove_life’. Si el número de vidas es de cero, se emite otro evento que vamos a registrar con el nombre de ‘game_over’. EL personaje cambia de color. Se espera unos 300 milisegundos y tras ese tiempo vuelve a su color original y se pasa la variable “this.hitDelay” a “false” para que pueda volver a chocar con el personaje.

Añadimos en la colisión del personaje con la estrella que cada vez que recoja una, se cree otra bomba:

```
//Colisión entre el personaje y las estrellas
        this.physics.add.overlap(this.personaje, this.star, () =>{
        this.sound.play('pop');
        this.registry.events.emit('update_points');
        this.star.destroyItem();
        this.bombGroup.addBomb();
        });
```

Marcador de puntos:

Vamos a crear en la carpeta “scenes” el archivo “UI.js”:

```
class UI extends Phaser.Scene{
    constructor(){
        super({key: 'UI'});
    }
    init(){
       console.log('Se inició la escena UI');

    }
    create(){

    }
}
export default UI;
```

Para lanzar esta escena creamos en la escena “Play.js” creamos la función “init()”:

```
init(){
        this.scene.launch('UI');
    }
```

Para que se reconozca la escena hay que importarla en el archivo “”main.js” y añadirla en el parámetro “scene”:

```
import UI from './scenes/UI.js';
```

```
    scene: [
        Bootloader,
        Play,
        UI
    ]
```

En el archivo “UI.js”, el método “init()” se programa la creación de los corazones que representan las vidas y el marcador de puntos:

```
init(){
        console.log('Se inició la escena UI');
        this.actual_point = 0;

    }
```

Y en la función o método “create()”:

```
create(){
        this.groupLife = this.add.group({
            key: 'life',
            repeat: 2,
            setXY: {
                x: 50,
                y: 30,
                stepX: 25
            }
        });
         //Marcador con puntos
         this.points = this.add.bitmapText(
            this.scale.width -40,
            20,
            'pixelFont',
            Phaser.Utils.String.Pad('0', 6, '0', 1)
        ).setOrigin(1, 0).setTint(0x000000);

        //Eventos
        this.registry.events.on('remove_life', () => {
            this.groupLife.getChildren()[this.groupLife.getChildren().length -
 1].destroy();
        });
        //Evento game_over
        this.registry.events.on('game_over', () => {
            this.registry.events.removeAllListeners();
            this.scene.start('Menu', {points: this.actual_point});
        });
        //Sumar puntos
        this.registry.events.on('update_points', () => {
            this.actual_point += 10;
            this.points.setText(Phaser.Utils.String.Pad(this.actual_point, 6, '0', 1));
        });
    }
```

Escena “Menu”:

Vamos a ir al archivo “Bootloader.js” y en lugar de iniciar la escena “Play” al completarse la carga vamos a cargar la escena “Menu” y también vamos a reproducir el sonido ambiental del juego:

```
//Sonido ambiental
  this.sound.play('bongo', {loop: true});
  //Lanzo la escena Menu
  this.scene.start('Menu');
```

Como aún no tenemos configurado la escena “Menu” no se ve aún nada. Vamos a configurarla. Para empezar cargo las imágenes en el método “create”. Copio de la escena Play la carga de las paredes y el suelo:

```
class Menu extends Phaser.Scene{
    constructor() {
        super('Menu');
    }
    create(){
        //Añadimos Imágenes de plataforma y cielo
        this.add.image(0, 0, 'sky').setOrigin(0);
```

```
        this.add.image(400, 568, 'ground').setScale(2);
        this.add.image(600, 400, 'ground');
        this.add.image(50, 250, 'ground');
        this.add.image(750, 220, 'ground');
        this.add.image(this.scale.width -15, 0, 'wall')
            .setOrigin(0)
            .setScale(.5, 5)
        this.add.image(0, 0, 'wall')
        .setOrigin(0)
        .setScale(.5, 5)
        //Añadimos logo
        this.logoMenu = this.add.image(
            this.scale.width/2,
            this.scale.height/2,
            'logo'
        ).setScale(4).setInteractive();

        //Añadimos texto
        this.titulo = this.add.bitmapText(this.scale.width/2, 100, 'pixelFont',
            'EXTRAVEGA').setTint(0xff0000).setScale(4).setOrigin(0.5);

                //Escribimos los puntos con texto
                this.pointsText = this.add.bitmapText(
                    this.scale.width/2,
                    this.scale.height - 100,
                    'pixelFont',
                    'PUNTOS ' + this.points
                ).setDepth(2).setOrigin(.5).setScale(2).setTint(0xff0000);

                //Incluyo el texto para la mejor puntuación
                this.bestPointsText = this.add.bitmapText(
                    this.scale.width/2,
                    this.scale.height - 150,
                    'pixelFont',
                    'MEJOR ' + this.bestPoint
                ).setDepth(2).setOrigin(.5).setScale(2).setTint(0xff0000);

                //Inicio el tween de salida del menú
                this.logoMenu.on(Phaser.Input.Events.POINTER_DOWN, () => {
                    this.add.tween({
                        targets: this.logoMenu,
                        ease: 'Bounce.easeIn',
                        y: -200,
                        duration: 1000,
                        onComplete: () => {
                            this.scene.start('Play');
                        }
                    });
                    this.add.tween({
                        targets: [this.pointsText, this.bestPointsText],
                        ease: 'Bounce.easeIn',
                        y: 400,
                        duration: 1000,
                    });
                });

                if(this.points > this.bestPoint){
                    localStorage.setItem('best_point', this.points);
                }
    }
}
```

```
export default Menu;
```

Nota: Este menú se debe importar en el archivo “main.js” e incluir en el atributo “scene” de la configuración del juego:

```
import Menu from './scenes/Menu.js';
```

```
scene: [
        Bootloader,
        Play,
        UI,
        Menu
    ]
```

NOTA: Es importante el orden de las escenas para que se vean correctamente.

Vamos al método “init” de la escena “Menu” y iniciamos una variable para los puntos con valor inicial de “0” ➔

```
init() {
    //console.log('Se ha iniciado la escena Menu');
    this.points = 0;
}
```

Vamos a modificar el método “init” de la clase “Menu” para que reciba datos de otras escenas. Para eso, introducimos este parámetro y el siguiente código:

```
init(data) {
    //console.log('Se ha iniciado la escena Menu');
    this.points = 0;
    //Comprobamos si se enviaron datos desde otras escenas
    if(Object.keys(data).length !== 0){
        this.points = data.points;
    }
}
```

Con “Object.keys(data).lenght” comprobamos cuántos datos se enviaron y si es distinto de “0” es porque se envió alguno (hay que comprobar que se envió alguno para evitar errores)

En la escena “UI” en el evento ‘game_over’ añadimos el envío de puntos:

```
//Evento game_over
this.registry.events.on('game_over', () => {
    this.registry.events.removeAllListeners();
    this.scene.start('Menu', {points: this.actual_point});
});
```

```
create() {

    const pointDB = localStorage.getItem('best_point');
    this.bestPoint = (pointDB !== null) ? pointDB : 0;
```

En la escena ”Menu”, en el “create” añadimos para la mejor puntuación:

```
//Incluyo el texto para la mejor puntuación
this.bestPointsText = this.add.bitmapText(
    this.scale.width/2,
    this.scale.height - 80,
    'pixelFont',
    'MEJOR ' + this.bestPoint
).setDepth(2).setOrigin(.5);
```

Debajo del código donde dibujamos el texto con los puntos, escribimos:

Es un código similar pero cambiamos el texto por “MEJOR” y la variable a concatenar por “this.bestPoint” ➔

Tras el inicio del “tween” de salida del menú establezco la lógica para la mejor puntuación:

```
if(this.points > this.bestPoint){
    localStorage.setItem('best_point', this.points);
}
```

Añado el tween para las dos puntuaciones:

```
//Inicio el tween de salida del menú
this.logoMenu.on(Phaser.Input.Events.POINTER_DOWN, () =>
    this.add.tween({
        targets: this.logoMenu,
        ease: 'Bounce.easeIn',
        y: -200,
        duration: 1000,
        onComplete: () => {
            this.scene.start('Play');
        }
    });
    this.add.tween({
        targets: [this.pointsText, this.bestPointsText],
        ease: 'Bounce.easeIn',
        y: 400,
        duration: 1000,
    });
});
```

Con esto hemos completado nuestro video juego. Espero que te haya gustado.