Archivos imprescindibles en un juego con phaser:

boot.js:

1- Con setPath podemos establecer el prefijo que se añadirá a todos los load que aparecen a continuación -> `this.load.setPath('assets/sprites/');`

2- Ahora cargamos individualmente cada archivo:

```
this.load.image('platform', 'platform.png');
this.load.image('base', 'base.png');
```

3- Al acabar de cargar assets, en create, cambiamos a la escena con la que comienza nuestro juego:

```
create() {
    this.scene.start('level');
  }
```

game.js:

1- Importamos todos aquellos archivos que se vayan a tener que cargar como escenas: los niveles, habitaciones, pantallas. NO hay que importar objetos o elementos del escenario.

2- Creamos una configuración para el juego (ver doc. para más parámetros de configuración). Algunos parámetros básicos que podemos definir son:

```
type: Phaser.AUTO,
width:  1000,
height: 500,
scale: {
    // mode: Phaser.Scale.FIT,
    autoCenter: Phaser.Scale.CENTER_HORIZONTALLY
},
pixelArt: true, // Prevent pixel art from becoming blurred when scaled
scene: [Boot, Level, End], // Todas las escenas que importamos antes
                           // Boot va la primera ¡!
physics: {
    default: 'arcade',
    arcade: {
        gravity: { y: 400 },
        debug: true
    }
}
```

3- Creamos el nuevo juego con `new Phaser.Game(config);`

Documentación **Arcade**: https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Arcade.ArcadePhysics.html

Documentación **Matter.js**: https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Matter.MatterPhysics.html

Physics:

`default: (x nombre del motor)`

`x: {definimos sus parámetros}`

Ejemplo de cómo hacer una barra de progreso: https://gamedevacademy.org/creating-a-preloading-screen-in-phaser-3/

Escenas de juego:

1- `export default class Level extends Phaser.Scene` -> extender aquello que sea necesario. Aquí entra la orientación a objetos y las herencias.
2- **Constructor**: Siempre debemos llamar al super en el constructor.
   a. Si extiende a Phaser.Scene -> pasar la key de la escena que estamos creando: `super({ key: 'level' });`
   b. Si extiende a Phaser.GameObjects.Sprite -> pasar la escena, posiciones x e y, la textura y, opcionalmente el frame (mirar doc).

Estos son los más comunes, pero podrían extender otra cosa. Deberíamos mirar la documentación y ver que parámetros hay que pasarle.

3- **Preload**: en principio, al usar boot, no haría falta. Pero es aquello que se hace la primera vez que se añade la escena.
4- **Init**: Sirve para pasar datos desde fuera cuando creamos esta escena.
5- **Create**: se ejecuta cuando llamamos a la escena (hacemos un start).
6- **Update**.
7- **Preupdate**. Update que usamos para objetos, en lugar de escenas.
8- Resto de métodos propios que vayamos a querer usar en esta escena.

Creación de objetos:

1- `export default class Player extends Phaser.GameObjects.Sprite`
2- Llamar al super en el constructor.
3- Crear/inicializar las variables de clase que queramos.
4- `this.scene.add.existing(this);`
   `this.scene.physics.add.existing(this);`
5- `this.scene.physics.add.collider(this, player);` si queremos que tenga un collider.
6- `this.body.setCollideWorldBounds();` si queremos que no se salga de los límites.
7- Declarar valores para las físicas: speed, jumpSpeed, etc.
8- `this.cursors = this.scene.input.keyboard.createCursorKeys();` para declarar input. Si queremos usar otras teclas: `console.log(Phaser.Input.Keyboard.KeyCodes)` nos da la lista de nombres de las teclas. También se puede mirar aquí. Haríamos:
   `keyX = this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.X);`
9- En los métodos propios de phaser, llamar siempre al **super**.
10- `this.scene.physics.add.existing(this, true);` true como parámetro para estáticos

Crear un objeto como parte de un grupo:

Al hacer esto no tendríamos por qué declarar el objeto en una variable, si no indicar en el constructor a qué grupo pertenece, y eso sí haber declarado ese grupo antes. En el propio constructor del objeto, que recibe ese grupo como parámetro, le hace `group.add(this)`. Recordar además hacer el punto 4 del apartado anterior: añadir el objeto a la escena y añadir sus físicas.

Para llamar una función de todos los miembros de un grupo:

```
this.enemies.children.each(function (enemy) {
            enemy.stopMoving();
        }, this);
```

Colisiones entre objetos para que no se atraviesen:

Ambos deben haber hecho el paso 4 en creación de objetos, y además uno de ellos debe tener `this.scene.physics.add.collider(this, player);` , siendo player el objeto con el que queremos que choque.

En caso de querer comprobar un overlap simplemente haríamos

`(this.scene.physics.overlap(this.scene.player, this)`

Sin tener que añadir collider a ninguno de los dos objetos.

Notas generales de programación:

1- metodo(parametro = null) // al hacer esto, si recibe un parámetro lo toma, pero si no, asume null (o cualquier otra cosa).
2- array.**filter**(o => o !== a) // elimina del vector el elemento a -> deja todo lo distinto de a.
3- Asegurarse de no gestionar las colisiones dos veces. (Repetir gestión de colisiones).
4- Para saber si una tecla está siendo **pulsada**: .isDown. Ejemplos:
   `this.cursors.left.isDown` donde cursors es una variable en la que guardamos `this.scene.input.keyboard.createCursorKeys()`.
   Otra forma es haciendo lo mismo pero en la variable guardando una tecla concreta. Haríamos `this.key.isDown`.
5- Un ejemplo de un **listener** de evento: `this.input.keyboard.on('keydown', function (event) { this.scene.start('level'); }, this);`
6- Los códigos para pulsar una tecla son: `'keydown/keyup-SPACE/A/B…'` `(keydown-SPACE)`
7- https://rexrainbow.github.io/phaser3-rex-notes/docs/site/timer/ Doc sobre el timer y sus métodos (ver tiempo restante, ver tiempo que ha pasado etc).
8- `this.cameras.main.startFollow(this.player);` Para que la cámara te siga.
9- `.setScrollFactor(0);` hace que eso a lo que se aplica se mueva 'pegado' a la cámara. Si lo ponemos a 1, se queda en el sitio.
10- `this.cameras.main.setBounds(0, 0, 1600, 400);` Para hacer un escenario de juego más grande de lo que se ve al principio.
11- `this.cameras.main.setBounds(0, 0, sizeX, sizeY);` Para redimensionar la cámara.
12- `getBottomLeft/Right()` sobre una imagen.
13- `function (event) {}, this);` es como `() => {}` pero le pasamos el this para indicar dónde se hace la función.
14- setDepth → para las capas de las imágenes. Si es un objeto como player/enemy poner en su constructor. Para elementos de la UI puedo ponerlo en el constructor de la escena.
15- `this.scene.game.config.physics.arcade.gravity.y;` // ACCESO A LA GRAVEDAD
16- https://rexrainbow.github.io/phaser3-rex-notes/docs/site/scenemanager/ Scene Manager

Textos en la escena:

https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Text.html

https://photonstorm.github.io/phaser3-docs/Phaser.Types.GameObjects.Text.html#.TextStyle