GUIDA PHASER

Introduzione… che cos'è Phaser?

Phaser è un game **framework** open source basato su *HTML5* e *Javascript*, e serve per lo sviluppo di veri e propri videogiochi. Il vantaggio di Phaser è che è gratuito e grazie alla sua vasta libreria che ricopre una buona parte della fisica dei videogiochi, ti permette di creare un videogioco completo a tutti gli effetti.

Ma come funziona Phaser?

La configurazione iniziale

Dopo aver creato la cartella con all'interno i due file **index.html** e **gioco.js** e aver aperto la cartella con *VisualStudioCode*, possiamo incominciare.
Il cuore di un gioco programmato in Phaser è la **configurazione iniziale**, in cui viene indicata la larghezza (*width*) e l'altezza (*height*) della schermata di gioco, la gravità, il debug (*true/false*) e le **3 scene** principali: *precaricamento*, *creazione* e *aggiornamento*.

```
var config = {
    type: Phaser.AUTO,
    width: 800,
    height: 600,
    physics: {
        default: 'arcade',
        arcade: {
            gravity: {
                y: 315
            },
            debug: false
        }
    },
    scene: {
        preload: preload,
        create: create,
        update: update
    }
};
```

Infine, per inizializzare la configurazione e far partire il gioco basterà scrivere questa riga di codice…

```
let game = new Phaser.Game(config);
```

Le 3 fasi: preload, create, update

Il precaricamento (**preload**) serve a caricare in precedenza le *immagini* che ci serviranno.

```
preload(){
    this.load.image('TileMap', './Assets/Sfondo.png');
    this.load.image('omino', './Assets/Omino.png');
    this.load.image('star', './Assets/star.png');
    this.load.any ('serpente', './Assets/serpente.png');
    this.load.image('completed', './Assets/completed.jpg');
    this.load.image('bianco', './Assets/bianco.jpg');
}
```

La fase della creazione (**create**) serve a gestire lo *sprite*, i *nemici*, le *collisioni*, le *vite*, i *punteggi*, le *vittorie* e i *gameover* del nostro gioco.

```
this.physics.add.collider(this.omino,layer);
this.physics.add.collider(serpente,layer);
this.physics.add.collider(serpente,this.omino);
```

```
this.physics.add.overlap(this.omino, serpente, () => {
    this.scene.start("GameOver");
});
```

Nell'ultima fase, l'aggiornamento (**update**), solitamente si gestiscono i *tasti* per far muovere l'omino. L'update è un ciclo infinito.

```
if(this.tastiera.left.isDown){
    this.omino.setVelocityX(-320);
    this.omino.setFlipX(true)
}
else if(this.tastiera.right.isDown){
    this.omino.setVelocityX(320);
    this.omino.setFlipX(false)
}
else{
    this.omino.setVelocityX(0);
}
if((this.tastiera.up.isDown) && this.omino.body.blocked.down){
    this.omino.setVelocityY(-330);
}
```

L'omino

Per giocare ci serve innanzitutto un **personaggio**. In Phaser è un'immagine caricata come sprite: in questo modo esso ha un *corpo*, una *posizione*, una *fisica* e si può (deve!) anche spostare. Carichiamo l'immagine con il seguente comando: ***this.load.image***.

```
this.load.image('omino', './Assets/Omino.png');
```

'omino' → è il nome che diamo all'immagine.
./Assets/Omino.png → è il percorso dell'immagine.

Successivamente lo facciamo diventare uno **sprite** con questo comando: ***this.physics.add.sprite***.

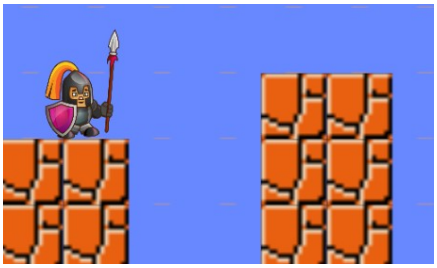
```
this.omino = this.physics.add.sprite(100, 600, 'omino').setScale(0.25).setDepth(1);
```

this.omino → è la variabile. Il this. davanti ci fa evitare di dichiararla. Senza il this. davanti avremmo dovuto dichiararla in questo modo: var omino; .
(100) → è la posizione sull'asse X.
(600) → è la posizione sull'asse Y.
'omino' → è il nome che abbiamo dato in precedenza all'immagine.
.setScale → serve a regolare la grandezza dell'immagine, in questo caso abbiamo impostato 0.25.
.setDepth → serve a determinare la profondità dell'immagine, che in questo caso è 1.

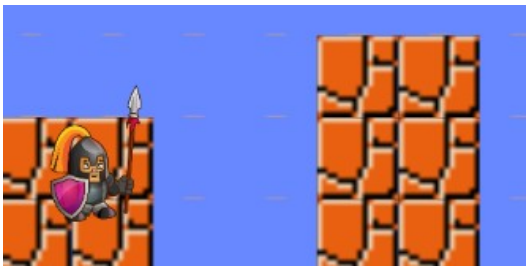
Le collisioni

Le **collisioni** sono fondamentali in ogni videogioco. Senza collisioni nessun gioco può funzionare. E' così anche in un semplice videogioco in Phaser: ad esempio, senza collisioni, il nostro omino cadrebbe e attraverserebbe il terreno uscendo dalla schermata di gioco.

Con le collisioni…



Senza collisioni…



Per far si che l'omino non attraversi le piattaforme è sufficiente usare il comando ***this.physics.add.collider***, utilizzando come 2 parametri la variabile dell'omino e del pavimento.

```
this.physics.add.collider(this.omino,layer);
```

I tasti

Ovviamente per giocare dobbiamo far si che il nostro omino si possa muovere. I tasti base utilizzabili sono: **freccia su-giù-destra-sinistra** e la **barra spaziatrice**. Tutto questo si gestisce nell'*update*. Come possiamo vedere nelle righe di codice dell'immagine sottostante, quando verrà premuta la freccia sinistra l'omino si muoverà sull'asse X a sinistra (stessa cosa per farlo muovere a destra e per farlo saltare).

```
if(this.tastiera.left.isDown){
    this.omino.setVelocityX(-320);
    this.omino.setFlipX(true)
}
```

Nemici e ostacoli

Nemici e **ostacoli** rappresentano un'altra delle tante parti fondamentali per la creazione di un videogioco. Grazie ad essi il gioco diventa più complicato ma anche più divertente. In Phaser sono delle semplici immagini caricate come **sprite** (come anche il nostro omino). Hanno un *corpo*, una *posizione*, una *fisica* e possono anche *spostarsi* e *affliggere danno* al nostro omino.

Come far muovere i nemici

Per far spostare i nemici da una parte all'altra della mappa si utilizza il comando ***this.tweens.add***. Grazie ad esso possiamo gestire lo spostamento (*value*) e la velocità (*duration*) con cui il nostro nemico si muoverà.

```
this.tweens.add({
    targets: serpente,
    props: {
        x: {
            value: 1100,
            duration: 2000,
            flipX: true
        },
    },
    ease: 'Linear',
    yoyo: true,
    repeat: -1
});
```

Le scene

Le **scene** in Phaser sono molto utili e permettono di gestire ad esempio i *livelli*, il *game over* e la *vittoria* del gioco. Solitamente si crea il file **config.js** in cui vengono caricare e importate le varie scene. In questo modo possiamo creare un file per livello (*es. Livello1.js per il livello 1, Livello2.js per il livello 2 e così via…*) e collegarlo al config.js. Nel file config.js, oltre alle scene, dobbiamo inserire anche la configurazione iniziale e la riga di codice per lo start del gioco.

```
import Livello1 from "./Livello1.js";
import Livello2 from "./Livello2.js";
import Livello3 from "./Livello3.js";
import Livello4 from "./Livello4.js";
import Livello5 from "./Livello5.js";
import GameOver from "./GameOver.js";
import Completed from "./Completed.js";
import Win from "./Win.js";
let config = {
    width: 1000,
    height: 550,
    parent: 'inizio',
    backgroundColor: 0x6888ff,
    physics:{
        default:"arcade",
        arcade:{gravity:{y:500},debug:false},

    },
    scene: [Livello1, Livello2, Livello3, Livello4, Livello5, GameOver, Win, Completed]
};

let game = new Phaser.Game(config);
```

import Livello1 from "./Livello1.js"; → con questo comando si importa il file Livello1.js nel file config.js

scene: [Livello1, Livello2, Livello3, Livello4, Livello5, GameOver, Win, Completed] → qui si inseriscono i nomi delle scene nell'ordine in cui devono comparire.

Ogni file collegato al config.js dovrà avere questa struttura.

```
class Livello1 extends Phaser.Scene{
  constructor(){
    super("Livello1")
  }
  preload(){}
  create(){}
  update(time, delta){}
}
export default Livello1;
```

Vittoria e GameOver

Con le scene è molto semplice gestire il **gameover** e la **vittoria** del gioco. Nel gioco che ho creato ho fatto in questo modo. Ho creato 2 scene: *Win.js* e *GameOver.js*.

Win.js

```
class Win extends Phaser.Scene{
    constructor(){
        super("Win")
    }

    preload(){
        this.load.image('win', './Assets/win.png');
        this.load.image('bianco', './Assets/bianco.jpg');
    }

    create(){
        var win;
        win=this.add.image(500, 274, 'win').setDepth(50).setVisible(true).setScale(2);
        this.sfondo = this.add.image(500, 274, 'bianco').setDepth(20).setScale(3.5);
    }

    update(time, delta){
    }

}

export default Win;
```

Per la vittoria, quando l'omino, nell'ultimo livello, prende la stella (ovvero quando entra in collisione con essa) compare la scena **Win.js** (un'immagine con scritto VICTORY). Sono bastate 3 righe di codice per fare ciò: ho utilizzato il comando ***this.physics.add.overlap*** e per far partire la scena ho utilizzato ***this.scene.start***.

```
this.physics.add.overlap(this.omino, stars, () => {
    this.scene.start("Win");
});
```

GameOver.js

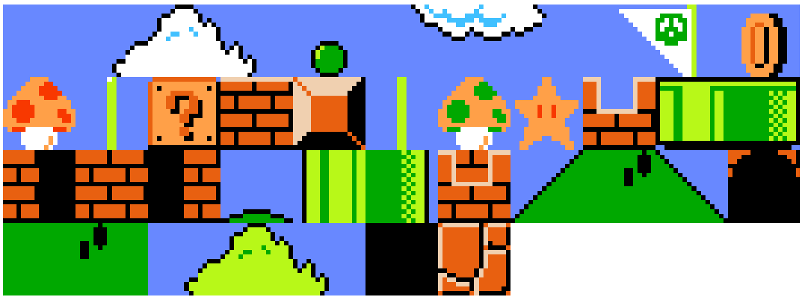
```
class GameOver extends Phaser.Scene{
    constructor(){
        super("GameOver")
    }

    preload(){
        this.load.image('gameover', './Assets/gameover.png');
        this.load.image('bianco', './Assets/bianco.png');
    }

    create(){
        var gameover;
        gameover=this.add.image(500, 274, 'gameover').setDepth(50).setVisible(true).setScale(1.2);
        this.sfondo = this.add.image(500, 274, 'bianco').setDepth(20).setScale(3.5);
    }

    update(time, delta){
    }

}

export default GameOver;
```

Per il gameover invece, quando l'omino entra in contatto con i nemici o gli ostacoli parte la scena **GameOver.js** in cui è caricata l'immagine del Game Over. Ho utilizzato anche questa volta il comando ***this.physics.add.overlap*** e per far partire la scena ***this.scene.start***.

```
this.physics.add.overlap(this.omino, serpente, () => {
    this.scene.start("GameOver");
});
```

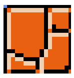
Tile Map

Una **tilemap** non è altro che un’immagine composta da varie mattonelle, chiamate *tiles*. Con le tilemap possiamo creare delle mappe risparmiando spazio e risorse. In Phaser si può implementare e pensare come una **matrice**: ad ogni mattonella dell’immagine corrisponde infatti un numero (0,1,2,3…) e inserendo nelle posizioni che vogliamo questi numeri possiamo creare la mappa. Successivamente si deve anche gestire il *movimento della mappa* e la *collisione dei blocchi* con gli altri oggetti nella mappa.

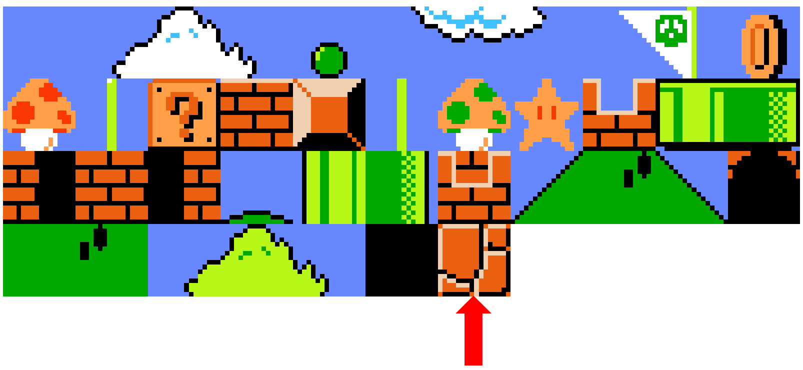


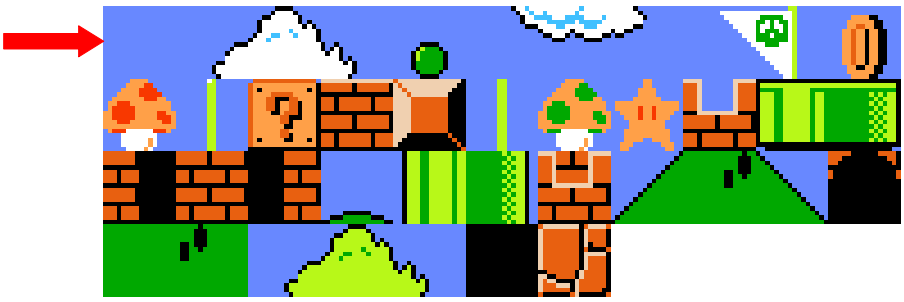
```
[ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
[ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
[ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
[ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
[ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
[ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
[ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
[ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,0,0,0,0,0,0,0,0 ],
[ 0,0,0,0,39,39,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,39,39,0,0,0,0,0,0,0 ],
[ 39,39,0,0,39,39,0,0,39,39,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,39,39,39,39,0,0,0,0,0,0 ],
[ 39,39,0,0,39,39,0,0,39,39,0,0,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39 ]
```

Esempio pratico

Mettiamo caso che vogliamo creare delle piattaforme in una mappa con questo blocco → 

Nell’immagine questo blocco si trova nella mattonella **39**





Il cielo invece vogliamo rappresentarlo con questo blocco → 🟦

Nell’immagine questo blocco si trova nella mattonella **0**

Nel nostro file .js ci basterà allora creare una variabile e chiamarla ad esempio level e all’interno creare la nostra matrice. Dopo inseriamo il numero **39** dove vogliamo per creare le nostre *piattaforme* e lo **0** per il *cielo*.

```
const level = [
    [ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
    [ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
    [ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
    [ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
    [ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
    [ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
    [ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ],
    [ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,0,0,0,0,0,0,0,0 ],
    [ 0,0,0,0,39,39,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,39,39,0,0,0,0,0,0,0 ],
    [ 39,39,0,0,39,39,0,0,39,39,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,39,39,39,39,0,0,0,0,0,0 ],
    [ 39,39,0,0,39,39,0,0,39,39,0,0,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39,39 ]
];
```

Il risultato sarà questo…